Due: **Monday September 23rd by 8:00 a.m., via Blackboard.** Optional hard copy may be turned in during lab.

**Overall Assignment**

This assignment extends HW1 to include colored lights shining on colored objects, which will require some conversions between data types as well as additional calculations. We will also convert the final resulting color to a gray-scale value and report the results in some different formats.

**Background: Computer Graphics**

Recall that in HW1 we calculated the gray-scale intensity of a light shining off of a single polygon using the formula

\[ I = I_aK_a + I_dK_d\cos(\theta) \]

where \(I_a\) and \(I_d\) were single-valued light intensities in the range 0 to 255 and \(K_a\) and \(K_d\) were single-valued reflectivity coefficients in the range 0.0 to 1.0.

For this assignment we will work in color for both the light intensities and the reflectivities, which means that \(I_a\), \(I_d\), \(K_a\), and \(K_d\) will each have red, green, and blue components, initially read in as integers in the range 0 to 255 and then converted to doubles for internal calculations, with the final result converted back to integers for reporting purposes.

**Background: Computer Color**

Colors in computer graphics are often expressed in hexadecimal format, in the form 0xRRGGBB, where the digits RR indicate the amount of red present, from a low of 00 (zero) to a high of FF (255 in decimal), and GG and BB indicate the amounts of green and blue respectively. The following table shows some of the standard colors and their RGB values:

<table>
<thead>
<tr>
<th>Color</th>
<th>Value</th>
<th>Value</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black</td>
<td>0x000000</td>
<td>0x808080</td>
<td>50% Gray</td>
</tr>
<tr>
<td>White</td>
<td>0xFFFFFF</td>
<td>0x404040</td>
<td>25% Gray</td>
</tr>
<tr>
<td>Red</td>
<td>0xFF0000</td>
<td>0xFF8080</td>
<td>Pink</td>
</tr>
<tr>
<td>Green</td>
<td>0x00FF00</td>
<td>0x008000</td>
<td>Dark Green</td>
</tr>
<tr>
<td>Blue</td>
<td>0x0000FF</td>
<td>0x8080FF</td>
<td>Light Blue</td>
</tr>
<tr>
<td>Cyan</td>
<td>0x00FFFF</td>
<td>0xFF8000</td>
<td>Orange</td>
</tr>
<tr>
<td>Magenta</td>
<td>0xFF00FF</td>
<td>0xC00000</td>
<td>Maroon</td>
</tr>
<tr>
<td>Yellow</td>
<td>0xFFFF00</td>
<td>0x400000</td>
<td>Brown</td>
</tr>
</tbody>
</table>
Background: Data Conversions

For this assignment there are three important data conversions that will be needed:

1. **Extraction of red, green, and blue components from a 32-bit int:** Although we express RGB colors notationally using hexadecimal format, internally we will still be reading in a single 32-bit int, were the low-order 8 bits hold the blue color information, the next 8 bits the green color information, and the next 8 bits the red color information. (The remaining high-order 8 bits typically hold transparency information, which we will be ignoring for this assignment.)

The low-order part of any number can be extracted from the rest of the number using the mod operator, to determine the remainder after integer division. In this case we want to extract the low-order 8 bits, which is the remainder in the range of 0 to 255 after dividing by 256, so if redGreenBlue is an integer, then redGreenBlue % 256 will also be an integer, composed of the low-order 8 bits of redGreenBlue. (I.e. the blue component.)

The easy way to extract the high-order bits of a number is to use integer division, and take advantage of the fact that any fractional components will be truncated. So again if redGreenBlue is an integer containing full-color information, then redGreenBlue / 256 will be the original number "shifted right" by 8 bits, i.e. the red and green colors only with the green shifted down to the low-order 8 bits and the red to the next 8 bits.

Following the above steps will separate the redGreenBlue color information into blue and redGreen, and then applying them again to redGreen will separate that into red and green.

2. **Compilation of red, green, and blue color information into a single RGB color integer:** This is just the reverse of the previous conversion, which is fortunately much easier. What we need to do is to shift the red bits left by 16 places, shift the green bits left by 8 places, and then add up the shifted red and green values to the unshifted blue value. So if red, green, and blue are integers in the range of 0 to 255, and noting that 256 squared is 65536, then the composite redGreenBlue can be calculated as:

   \[ \text{redGreenBlue} = \text{red} \times 65536 + \text{green} \times 256 + \text{blue} \]

3. **Conversion of full color information into gray scale:** The general approach for converting RGB colors to gray scale is to multiply each of the RGB components by a weighting factor and then adding up the result. However the obvious weighting factors of 0.33333333 each are not commonly used, because human eyesight is much more sensitive to shades of green, and least sensitive to shades of blue. There are actually many different gray scale conversions in use, with two of the more common being:

   \[ \text{NTSC gray} = 0.299 \ R + 0.587 \ G + 0.114 \ B \]
   \[ \text{HDTV gray} = 0.2126 \ R + 0.7152 \ G + 0.0722 \ B \]

Program Details

For this assignment you are to write a computer program that asks the user for ambient and diffuse lighting information and reflectivity coefficients as full-color RBG integers in the form 0xRRGGBB (and the angle for cosine lighting), and then calculates and reports the full-color
result of lighting a colored polygon with a colored light. The result should also be reported as gray scale using both the NTSC and HDTV conversions, with the gray scale results reported on a scale of 0.0 to 1.0 instead of 0 to 255.

- Your program should first print out your name and ACCC netID (e.g. jbell), and explain to the user what the program does.
- Your program should then ask the user to enter the ambient light intensity, diffuse light intensity, ambient reflection coefficient, diffuse reflection coefficient, and the angle between the normal of the polygon and the direction of the light source in degrees. The user should be asked to enter the light intensities and reflectivity coefficients as hexadecimal numbers of the form 0xRRGGBB as described here, which your program should scanf into unsigned int variables using the %x format specifier.
- The integers read in should then be converted to separate red, green, and blue doubles, using conversion type 1 described above. The reflectivity coefficients should be further normalized to the range of 0.0 to 1.0 by dividing the values read in by 255.
- Your program should then calculate the overall resulting intensity according to the equation used in HW1, performing calculations for the red, green, and blue components separately, and working with doubles. Report these values as intermediary results. Because we have not yet covered decisions, you are not responsible for detecting bad input that would cause the result to exceed the maximum allowable value of 255 for the red, green, and/or blue components.
- Calculate the composite integer RGB value of the result using conversion type 2 above, and print out the result using %d, and either %x or %X format specifiers. (Note that the conversion formula listed above as number 2 requires integers for its inputs.)
- Calculate the gray scale value using both the NTSC and HDTV formulas, and report the results using exactly 5 digits to the right of the decimal point. Report also the % difference between the two values, assuming the NTSC value is the "correct" value. (or more accurately the "original" value.)

**Program Output**

- User input should be echoed as both its original form (%x or %X with 6 digits), decimal integer, (%d), and transformed to components (%d for Ia and Id, %f with three digits right of the decimal point for Ka and Kd). Calculated results should be reported as individual colors in decimal with two digits right of the decimal point and as individual integers and a composite integer in both %d and %x or %X with 6 hex digits. All of this should be reported as complete sentences, such as:

  With an ambient light source of 0xAB2AC0 (11217600, = 171 red, 42 green, 192 blue) and a diffuse light source of 0x0F8C0D (1018893, = 15 red, 140 green, 13 blue) shining at an angle of 22.5 degrees on a surface with an ambient reflectivity coefficient of 0x00FFFF (65535, = 0(0.000) red, 255(1.000) green, 255(1.000) blue) and a diffuse reflectivity coefficient of 0x326496 (3302550, = 50(0.196) red, 100(0.392) green, 150(0.588) blue), the resulting overall light reflection is 0x025CC7 (2(2.72) red, 92(92.70) green, 199(199.06) blue), with an NTSC gray level of 77.92, and HDTV gray level of 81.25, and a % difference of 4.274%.

(You may want to reword that, and/or break it up into multiple sentences, but the numbers should be printed in all the formats shown here. Accuracy of this sample is not guaranteed.)
Special Notes:

- **You should work out some sample problems by hand before writing any computer code.** ([http://www.mathsisfun.com/binary-decimal-hexadecimal-converter.html](http://www.mathsisfun.com/binary-decimal-hexadecimal-converter.html) may help.)

- The basic assignment should not use any loops. You may use loops in your program only if you implement one of the optional enhancements that require looping.

- Since you haven’t learned to test things, you can assume that all of the user input is good.

What to Hand In:

- Your code, **including documentation**, should be handed in electronically on Blackboard.

- All files should be zipped together into a single file, whose name is comprised of your ACCC netID followed by the course number followed by the letters "HW", followed by the assignment number. (E.g. jbell107HW1.zip) The zip file should be handed in via Blackboard. ([Your TA may provide alternative submission instructions.])(Your TA may provide alternative submission instructions.)

- The intended audience for the documentation file is a general end user, who might want to use this program to perform some work. They do not get to see the inner workings of the code, and have not read the homework assignment. You can assume, however, that they are familiar with the problem domain (e.g. lighting and lighting types.)

- A secondary purpose of the documentation file is to make it as easy as possible for the grader to understand your program. If there is anything special the grader should know about your program, be sure to document it in the documentation file. In particular, if you do any of the optional enhancements, then you need to document what they are and anything special the TA needs to do to run your program and understand the results.

- If there are problems that you know your program cannot handle, it is best to document them as well, rather than have the TA wonder what is wrong with your program.

- Make sure that your name appears at the beginning of each of your files. Your program should also print this information when it runs.

Optional Enhancements:

It is course policy that students may go above and beyond what is called for in the base assignment if they wish. These optional enhancements will not raise any student’s score above 100 for any given assignment, but they may make up for points lost due to other reasons.

- Check the data entered to verify that it is valid.
  - RGB composite colors should lie in the range 0 to 0xFFFFFFFF.
  - Calculated composite intensities should lie in the range of 0 to 255 for each component. The best approach for this assignment is to "clamp" the numbers into this range, by converting any calculated results that are larger than 255 to be 255.
  - Angles 0 to 90 degrees are reasonable for this problem. Angles from 90 to 180 would indicate a light source “behind” the polygon, and other angles are unreasonable.

- Ask the user if they would like to solve additional problems, and if so, repeat until done.

- Learn how to use the <<, >>, &,, |, and ^ bit twiddling operators, and use them with appropriate "masks" to extract the red, green, and blue components from the 32-bit int and then combine the results at the end back into a 32-bit int.

- Other enhancements that you think of – Check with TA for acceptability.

**Note:** Optional enhancements that you already mastered in one HW should not receive a lot of credit if you just repeat them again for later HW.