22

Pointers & Graphics

his assignment, will show you how to implement editing operations on PNG and JPG graphics files. Inside the assignment folder, you'll find several files and three extra folders:

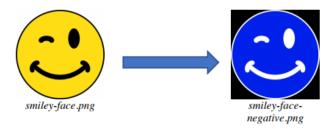
- The input folder contains the photos you're going to start with; the expected, folder contains the photos as they should look, and the actual folder will contain the photos after your filters have been applied. If your code fails one of the tests, the actual folder will also have a "diff" image you can examine. You can look at any of the photos just by double-clicking them in the IDE.
- I have compiled and linked the C libraries into a static library for this assignment. The header file includes prototypes for the library functions we are using. Note that these prototypes are wrapped in an **extern "C" { };** block to ensure our C++ programs can link to them. If we left this off, they wouldn't be found.
- The header file also contains the prototype for the negative() function. You will write your implementation in the implementation (.cpp) file. At the end of the implementation file, in the run() function, you'll find some sample code similar to that from the first part of the lesson. You can type <a href="mailto:mailto

The negative Function

In the **negative()** function you should change the image, passed as a **const pointer** to **unsigned char**, into one whose pixels are the **inverse** of those in the original image. There is nothing to return from the function.

To convert an image to its inverse, for each pixel, set all three of its red, green, and bLue values to be the inverse of their current color value. The inverse of a color value k is defined as 255 - k: the pixel (110, 52, 236) has an inverse of (145, 203, 19).

¹ Adapted from the <u>ImageShop assignment</u>, Stanford CS106A, Spring 2017



Look at the declaration for the **negative()** function:

```
*/
void negative(unsigned char * const img, int width, int height);
```

Notice the word **const** in the declaration of **img**. When the word **const** comes **after the star**, it means that value inside the pointer cannot be moved; you **cannot make it point to a different location**.

Why would the function want to prevent you from changing the value in the pointer? Because after the function is finished, the block of memory containing the image **must** be returned to the operating system. If your function was allowed to change that, the program would probably crash.

So, if **img** is a **const** pointer, then what should you do? Just **create a second pointer**, one that is not constant, that points to the same location, like this:

```
void negative(unsigned char * const img, int width, int height)
{
unsigned char * p = img; // can use p to change image
}
```

Now, you can **invert the first pixel** by using the following algorithm:

```
*p <- 255 - *p -> Invert the red component
p++ -> Move p to next component
*p <- 255 - *p -> Invert the green component
p++ -> Move p to next component
*p <- 255 - *p -> Invert the blue component
p++ -> Move p,
p++ -> skipping the alpha channel
```

How do you change the rest of the pixels? You need some type of loop. The best kind of loop for this is an **iterator loop**. Create a **second pointer** named **end**, one that points to the byte right past the end of the image, you could use a loop that looks like this:

HOMEWORK POINTERS & GRAPHICS

```
while (p != end)
{
    // Process element p points to
    p++;
}
```

The question is, how do you get this pointer variable named end?

Using Pointer Arithmetic

Applying the + and - operators to pointers is **pointer arithmetic**. Adding an integer to a pointer gives us a **new address value**. For each unit that is added to a pointer value, the internal numeric value must be increased by the size of the base type of the pointer. In our case, the base type of the pointer, (**unsigned char**) is 1. Here's the pseudocode:

```
Let p point to beginning of the image (p = img)
Let end be img + width * height * BPP (BPP given as 4)
While p != end
Invert each component and increment as above
```

You can use **auto** (type inference) to declare the variable **end**, so you don't have to manually write out the variable's type. Even though **img** is a **const** pointer, **end** is **not const** (just like **p** is not **const**). Make sure that **end** cannot be inadvertently moved, by adding an additional **const**.

Once you've added these changes, run **make test** and make submit to turn it in. If you have problems, shout out on the discussion board, or come by my office hours.