Tips for module 3's assignment

Understanding BMP

Before you get started, you'll likely want to look at the BMP files you are provided with in a hex editor. Try to use Bless because it comes up in the exercises, but not all repositories have a working version (you will know if that applies to you; the possible issues I've seen involve crashes and graphical glitches). If you can't find a working version of Bless I recommend Okteta, which should be available from your package manager.

You can open the BMP files to get a better sense of where everything is. Graphical hex editors usually have indexing on the side so you can more easily count the bytes of data you're looking at. Below, I've divided test2.bmp into header, DIB header, and pixel data. Try to map out the different fields in the headers based on the byte offset. In a real hex editor, you'll be able to click on the bytes to see their decoded values.

Header (bytes 0-13):

```
42 4D 46 00 00 00 00 00
00 00 36 00 00 00
```

DIB Header (bytes 14-53):

```
28 00 00 00 02 00 00 00
02 00 00 00 01 00 18 00
00 00 00 00 10 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
```

Pixel data (bytes 54+):

```
00 00 FF FF FF FF 00 00
FF 00 00 00 FF 00 00 00
```

Every 3 bytes makes a pixel; each row is 0-padded to a multiple of 4 bytes. In this example, the width is 2 pixels, which total to 6 bytes of data. To reach the next multiple of 4, 8 bytes, each row is padded with the bytes 00 00. Another thing to be aware of is that the BMP file format stores pixel data from the bottom row of the image to the top and every three bytes that correspond to a pixel are the blue, green, and red channels, respectively. For the images you're given, the channel values can be between 0 and 255 (inclusive).

When to allocate memory in the heap for a struct

If you're confused about when you do and don't need to use the malloc family of functions, consider their return type: a void pointer.* It is never going to make sense to write:

```
struct MyStruct x = malloc(sizeof(struct MyStruct));
```

You will likely be able to compile such a statement with a warning about implicit casting a pointer to not a pointer. (Speaking of which, read the compiler warnings! I know they look overwhelming, but they are written in English with code that you wrote interspersed, and they are almost always pointing out buggy code. So save yourself some time in the debugger and don't run until you've fixed the warnings.) When you allocate memory for a struct, it should be assigned to a POINTER to a struct, like so:

```
struct MyStruct *x = malloc(sizeof(struct MyStruct));
```

With your memory properly allocated, you are now free to assign the members of x without a seg fault.

That is not to say there is never a time when it is appropriate to declare simply `struct MyStruct x;` and start using the memory that the operating system has provided you on the stack right away, filling in all the members of x. But bear in mind that x, declared in that way, would only be a stack variable. If you do this inside of anything but main(), the memory assigned to x will get freed by the operating system once x goes out of scope. To prevent that from happening you need to use malloc().

*Fun fact: In C, you do not need to explicitly cast the void pointer returned by malloc(). If your IDE is giving you red squigglies when you don't do that, it's probably configured to check C++ syntax, so you may want to fix your settings! There's no harm in explicitly casting it to another type of pointer though, so do what makes the most sense to you.

Misc. Tips

- You know `int argc` and `char* argv[]`, the parameters typically passed to main()? `argc` is the number of command line arguments, including the call to the program itself (so there are actually `argc-1` arguments). `argv` is an array of the arguments themselves as strings (each entry is of type char pointer), so that `argv[0]` is the name of the program and the successive entries are the command line arguments in the order they were typed.

- The order you input the arguments shouldn't matter. A convenient way to do that in the unistd.h header is getopt(). Check the man page as well as this example, which shows how to access non-option arguments, flags, and flags with associated values: https://www.gnu.org/software/libc/manual/html_node/Example-of-Getopt.html

- The grayscale algorithm description is a little vague. After computing 0.299R + 0.587G + 0.114B for a given pixel, you need to set all that pixel's channels to that sum.

- Be careful of overflows and underflows when computing color shifts. Each channel maxes out at 255 and cannot go below 0 because we are using 8-bit color channels.

- General process for debugging that I recommend: Test one functionality at a time because it is annoying to not know which line caused a seg fault. If something doesn't work, put a breakpoint at the function that implements that functionality and step through the function, keeping an eye on the data with each step to make sure the contents of your structs are what you expect.

- Quick command to check for whether you made an exact copy of an image:

  ```
  diff [original] [copy]
  ```

  This does a bit-by-bit comparison of both files. If there is no output, the files have the exact same contents.