COM S 229 Project 1 Spring 2014

Part A. Assigned Monday, January 27th. Due Friday, February 21st, 11:59pm.

Part B. Assigned Monday, February 17th. Due Wednesday, March 12th, 11:59pm.

1 Project Summary (Part A)

You will build a C library containing data structures and procedures for loading, manipulating, and storing uncompressed images.

For Part A, you will create a C structure that contains all relevant data for an uncompessed bitmap using 32-bit color with RGBA (Red, Green, Blue Alpha) channels. You will also develop procedures for loading and storing these bitmaps in binary files, using a specified, custom format: SIMP. Finally, you will implement procedures for manipulating these bitmaps, and create executables that perform these operations:

- `crop picture.simp out.simp x y w h` — Load the bitmap from `picture.simp`, then crop the image by a rectangle with width w and height h starting at the position (x, y). Save the resulting image to `out.simp`.
- `bw pic.simp out.simp` — Load the bitmap `pic.simp` and create a new bitmap where the image is converted to black & white. Save the new grayscale image to `out.simp`.
- `colorshift pic.simp out.simp pattern` — Load the bitmap from `pic.simp` and perform a *color shift* according to `pattern`. Output the resulting image to `out.simp`.
- `overlay pic1.simp pic2.simp out.simp x y` — Load the two bitmaps from `pic1.simp` and `pic2.simp` and create a new bitmap given by drawing the second picture on top of the first picture using a translation such that the top-left corner of `pic2` is at the position (x, y) inside `pic1`. Use the Alpha channel to determine the amount of transparency to use.

Some more complicated tasks will be given in Part B, but these tasks will use combinations of the procedures required for the programs in Part A. Thus, it is *very important* that you design your data structures and procedures to be flexible to slight changes. In particular, each program should be associated with a C function that performs that task that is part of your library, and not embedded in the main method.

Since the SIMP format is not a standard format, there are executables `simp2bmp` and `bmp2simp` that can convert 16-bit BMP files to SIMP files, and vice-versa.

More specific information on these topics are presented in Section 6.

2 Project Goals

There are several learning objectives for this project. In earlier programming classes, all assignments were described in a detailed manner. One of our goals in this class is to keep the assignment slightly open-ended and underspecified. We want you to learn to think of possibilities and to ask for clarifications (to specify the problem better). Also, we want you to learn to handle complexity by dealing with slightly larger projects with more lines of code. We want to wean you from being too dependent on IDEs by being able to compile and run code from command line and to use makefiles. In addition, of course, we want to make sure you learn to use C.

In addition, you will demonstrate proficiency in the following C programming language features: structures, pointers, memory-management, (binary) file input/output, command-line arguments, source code organization, compilation, makefiles.

You may also find the following concepts helpful: unions, debuggers (such as `gdb`), memory error detectors (such as `valgrind`),

3 Tips on getting the project completed in time

- Start EARLY! Ask for help early!
- Set aside several times a week every week to make progress on the project.
- Focus on a part of the project and get that done.
- Write pseudo code and then write code.
- Test and make sure each small method is working.
- DO REMEMBER THAT LOTS OF UNEXPECTED THINGS HAPPEN AND CODING USUALLY TAKES MUCH MORE TIME THAN YOU EXPECT. SO START EARLY.

4 Grading

The following distribution of points will be used for this part of the project.

Category	**Points**
`crop`	50
`bw`	50
`colorshift`	50
`overlay`	50
`Makefile`	20
`README`	30
Total (Part A):	250

While each program is worth 50 points individually, if there are problems with common parts of the implementation (such as the file I/O) you will lose points for each program where the errors occur. Also, note that if your code does not build because of compiler or linker errors (either using make or by hand), you will likely lose much more than 20 points (if not all 200 points for the compiled programs).

Effectively: the TAs reserve the right to not grade your project at all if it fails to compile or link. **Test early, and test often** *on pyrite.*

5 A note on working together

This assignment is intended as an individual project. However, some amount of discussion with other students is expected and encouraged. The discussion below should help resolve the grey area of how much collaboration is too much.

Not allowed

Basically, any activity where a student is able to bypass intended work for the project, is not allowed. For example, the following are definitely not allowed.

- Working in a group (i.e., treating this as a group project).
- Posting or sharing code. DO NOT POST CODE.
- Discussing solutions at a level of detail where someone is likely to duplicate your code.
- Using a snippet of code found on the Internet, that implements part of the assignment. Generic code may be OK, but (for example) code that processes an bitmap file is definitely not OK. If you have any doubt, check with the instructor first.

As a general rule, if you cannot honestly say that the code is yours (including the ideas behind it), then you should not turn it in.

Allowed

- Sharing test files (please post them on Piazza).
- Discussions to clarify assignment requirements, file formats, etc.; again, please post these on Piazza.
- Highlevel problem solving (but be careful – this is a slippery slope).
- Contact the TAs or the instructor for help directly. However, our goal is to teach you to fish, so we will mostly be discussing STEPS to solving whatever problem you are facing.

If you are not sure whether doing something counts as academic dishonesty, then (1) DON'T DO IT and (2) ask a TA or the instructor.

6 Detailed Project Specification

Computers store images as numerical data in a fairly standard way. A rectangular shape is split into a number of identically-sized squares called *pixels*. An image's *resolution* is given by the width w and height h of the image in pixels. The coordinate of a pixel is given by a pair (x, y) where $x \in \{0, \dots, w-1\}$ and $y \in \{0, \dots, h-1\}$ and the coordinate $(0, 0)$ is the top-left corner and the coordinate $(w-1, h-1)$ is the bottom-right. The x coordinate measures the distance from the left edge while the y coordinate measures the distance from the top.

These pixels are associated with a color value and a transparency. The color value is given by three values: Red, Green, and Blue. For 32-bit color, these numbers are each specified by a single byte of data (8-bits, given as an *unsigned char*) with values from 0 to 255, where 0 has none of that color and 255 is full color. Thus, a pixel with (r, g, b) data being $(0, 0, 0)$ is black while $(255, 255, 255)$ is pure white. (Frequently, hexadecimal is used, so `#FF0010` specifies the color $(255, 0, 16)$.)

The fourth byte of data is the Alpha channel. This channel stores a number a, which specifies the pixel's opaqueness or transparency. A value of 0 says the pixel is entirely transparent, while a value of 255 is entirely opaque. This value is used to determine how much something "below" the image should affect what appears to the user. For mor information, see the `overlay` specification.

6.1 The SIMP Image Format

A file in SIMP format is a binary file with the following specification:

First, two 32-bit integers are given that specify the width and height of the image.

Then, the pixels are given in order as if reading a page: left-to-right then top-to-bottom (each row is contiguous).

RGBA data as 4 bytes per pixel, where the bytes are ordered as RGBA (red, green, blue, alpha).

All programs read and write to binary files using the SIMP file format, so it is a good idea to create methods for reading and writing an image that exist in a library outside of the main method.

Since standard image viewers will not be able to read SIMP files, you are supplied with two utilities: `bmp2simp` and `simp2bmp` that convert to-and-from 16-bit bitmap files[1] To take a BMP file `file.bmp` to SIMP format, use the command

```
bmp2simp file.bmp file.simp
```

To perform the opposite action, use the command

```
simp2bmp file.simp file.bmp
```

[1]See http://orion.math.iastate.edu/dstolee/229/project1/bmp2simp and http://orion.math.iastate.edu/dstolee/229/project1/simp2bmp.

The image-processing software GIMP (http://www.gimp.org/) can be used to take any standard image (including JPG, GIF, and PNG) and convert it to 16-Bit BMP using the Export feature. **Warning:** Unfortunately, `bmp2simp` cannot convert 32-bit BMP files!

6.2 Detailed Executable Specifications

We discuss below some of the more detailed specifications for the programs that are required. Keep in mind that not all error conditions or special cases are discussed explicitly. It is your responsibility to discover ambiguous requirements and to decide reasonable responses.

`crop`

The `crop` program performs a crop operation on an image, selecting a smaller portion to create a smaller image.

```
crop picture.simp out.simp x y w h
```

Load the bitmap from `picture.simp`, then crop the image by a rectangle with width w and height h starting at the position (x, y). Save the resulting image to `out.simp`. Do not change any pixel contents.

`bw`

The `bw` program converts an image to black and white.

```
bw pic.simp out.simp
```

Load the bitmap `pic.simp` and create a new bitmap where the image is converted to black & white. Each pixel is converted to black and white by averaging the Red, Green, and Blue values and then setting each color channel to that averaged value.

Save the new grayscale image to `out.simp`.

`colorshift`

The `colorshift` program performs an artistic color shift in the given image.

```
colorshift pic.simp out.simp pattern
```

Load the bitmap from `pic.simp` and perform a *color shift* according to `pattern`. The pattern should be one of the following strings:

```
RGB GBR BRG RBG BGR GRB RG GR RB BR GB BG
```

where a pattern, such as RGB says: send Red to Green, Green to Blue, and Blue to Red. (Observe that the last color is pushed to the first color.) The two-letter patterns swap two colors, such as RG to swap Red and Green, while keeping Blue the same. Output the resulting image to `out.simp`.

`overlay`

The `overlay` program takes two images and places one on top of the other.

```
overlay pic1.simp pic2.simp out.simp x y
```

Load the two bitmaps from `pic1.simp` and `pic2.simp` and create a new bitmap given by drawing the second picture on top of the first picture using a translation such that the top-left corner of `pic2` is at the position (x, y) inside `pic1`. If the Alpha channel is equal to 255 in a pixel of `pic2`, then the new pixel inherits all channel information from that pixel in `pic2`.

However, if the Alpha channel is between 0 and 254, then some of the pixel from `pic2` should be present. Suppose that (r_2, g_2, b_2, a_2) is the pixel from `pic2` to be placed on top of the pixel (r_1, g_1, b_1, a_1) from `pic1`. Then, calculate the new pixel (r, g, b, a) using the following formulas:

$$r = \frac{a_2}{255} r_2 + \frac{a_1(255 - a_2)}{255^2} r_1 \qquad g = \frac{a_2}{255} g_2 + \frac{a_1(255 - a_2)}{255^2} g_1$$

$$b = \frac{a_2}{255} b_2 + \frac{a_1(255 - a_2)}{255^2} b_1 \qquad a = 255 \left(\frac{a_2}{255} + \frac{a_1(255 - a_2)}{255^2} \right)$$

6.3 README

As part of your source-code deliverable, you will write a text file called `README` that contains a description of your project design. The `README` file should contain the following elements:

1. Your Name and NetID
2. A description of your data structure for storing an image.
3. A description of any extra details assumed due to ambiguous requirements in the project specification.
4. A file listing of the source code files, specifically mentioning the purpose of each code file.
5. A description of any extra features you added to the programs (or any extra programs you created).

6.4 Compilation and Makefile Requirements

All software must compile and run on `pyrite.cs.iastate.edu` without error. Simply typing `make` should build all of your executables. Also, `make clean` should remove the executables and any object files. You may include other targets for your convenience as you choose (e.g., "`make tarball`")

Your source code should be delivered as a gzipped tarball with the file name `netid.tar.gzip` (where `netid` is replaced by your NetID). Inside the tarball should be *source code only*, no compiled binaries. The grader will type the following commands to compile the software:

```
tar xvf netid.tar.gzip
make clean
make
cat README
```

Thus, your tarball should contain the `Makefile` in the root directory, and compile all binaries to that same directory. In addition, your makefile should use the `-ansi` and `-pedantic` flags when compiling via `gcc`.

If all of your source code is given in `.c` and `.h` files in one directory, you can compress your tarball using the command

```
tar czf netid.tar.gz *.c *.h Makefile
```

6.5 Submitting your work

You should turn in a gzipped tarball containing your source code, makefile, and a README file that documents your work. The tarball should be uploaded in Blackboard.

Your executables will be tested on `pyrite.cs.iastate.edu`. You should test early, and test often on pyrite. We will use some scripts to check your code; this means you should not change the name of the executables or the order of the command-line arguments.

Since all input in this project is given in the form of command-line arguments and by file input/output, your binaries should not output anything to the shell unless there is an error condition. Likewise, they should not take any input from standard in.

7 More Resources

http://en.wikipedia.org/wiki/Raster_image

http://en.wikipedia.org/wiki/Alpha_compositing

http://www.kevssite.com/2009/04/21/using-vi-as-a-hex-editor/