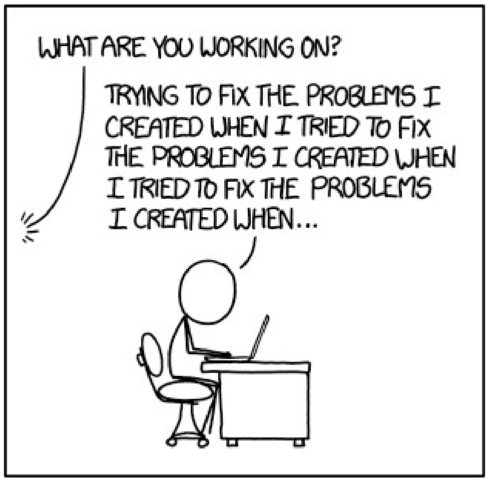
CS 218 – Assignment #11, Part A

Purpose: Become more familiar with operating system interaction, file input/output operations, and file I/O buffering.
Points: 250 (grading will include functionality, documentation, and coding style)

Assignment:

Write a program that performs some basic image manipulations. These manipulations include conversion to gray-scale, image brightening, or image darkening. The program will read the image manipulation option, source file, and output file from the command line. The command line qualifier will specify the image manipulation option ('-gr', '-br', or '-dk'). Next, the input file name and then the output file name. See below for an example. The program must perform basic error checking on the command line qualifiers.


WHAT ARE YOU WORKING ON?
TRYING TO FIX THE PROBLEMS I CREATED WHEN I TRIED TO FIX THE PROBLEMS I CREATED WHEN I TRIED TO FIX THE PROBLEMS I CREATED WHEN...

For example, to convert a color image to grayscale reading the file **`image0.bmp`** and create an output file named **`newImage.bmp`**, the program command line would look like:

```
ed-vm% ./imageCvt -gr image0.bmp newImage.bmp
```

The required procedures include the following:

- Function ***getArguments()*** to read and check the command line. The function should check the image option (must be '-gr', '-br', or '-dk'). The function should check the file names by attempting to open the files. If both files open, the function should return TRUE and the file descriptors. If there is an error, the function should display an appropriate error message (provided) and return FALSE.
- Function ***processHeader()*** to read and verify the header information. The function must check the signature, color depth, and bitmap size consistency. Refer to the BMP File Format section. If the information is correct, the entire header should be written to the output file, the width, height and file size should be returned to the main, and the function should return TRUE. If there is an error, the function should display an appropriate error message (provided) and return FALSE.
- Function ***getRow()*** should return one row of pixels. If a row can be returned, the function should return the row and return TRUE. If a row can not be returned because there is no more data, the function should return FALSE. If there is a read error, the function should dispaly an appropriate error message (provided) and return FALSE. To ensure overall efficiency, the function ***must*** perform buffered input with a buffer size of BUFF_SIZE (originally set to 1,000,000). *Note*, this will be changed for part B.
- Function ***writeRow()*** should write one row of pixels to the output file. If successful, the function should return TRUE. If there is a write error, the function should dispaly an appropriate error message (provided) and return FALSE. This function is not required to buffer the output. As such, each row may be written directly to the output file.
- The procedure ***imageCvtBW()*** should convert a row of pixels to gray scale using the provided algorithm.
- The procedure ***imageBrighten()*** should brighten a row of pixels using the provided algorithm.
- The procedure ***imageDarken()*** should darken a row of pixels using the provided algorithm.

Provided Main

The provided main program calls the various functions. ***The provided main program must not be changed in any way.*** All your functions must be in a separate, independently assembled source file. *Note*, to help with the initial program testing, it might be best to temporarily skip (comment out) the calls to the image manipulation routines.

Assemble and Linking Instructions

You will be provided a main (**`imageCvt.cpp`**) that calls the functions. Your functions should be in a separate file (**`a11procs.asm`**). The files will be assembled individually and linked together. When assembling, and linking the files for assignment #11, use the provided **makefile** to assemble, and link. *Note*, **only** the functions file, **`a11procs.asm`**, will be submitted. So, do not alter the provided main.

Debugging -> Command Line Arguments

When debugging a program that uses command line arguments, the command line arguments must be entered after the debugger has been started. The debugger is started normally (**ddd** <program>) and once the debugger comes up, the initial breakpoint can be set. Then, when you are ready to run the program, enter the command line arguments. This can be done either from the menu (Propgram -> Run) or on the GDB Console Window (at bottom) by typing **`run <commandLineArguments>`** at the (gdb) prompt (bottom window).

Buffering

Since many image files can be very large, it could be difficult to pre-allocate enough memory to hold an entire large image. Further, that memory would be mostly unused when processing smaller images. To address this, the program will perform *buffered input*. Specifically, the program should read a full buffer of BUFF_SIZE bytes (originally set to 1,000,000). *Note*, this will be changed for part B. From that large buffer, the ***getRow()*** function would return one row (width *3 bytes). The next call the ***getRow()*** function would return the next row. As such, the ***getRow()*** function must keep track of where it is in the buffer. When the buffer is depleted, the readRow() function must re-fill the buffer by reading BUFF_SIZE bytes from the file. Only after the last row has been returned, should the ***getRow()*** function return a FALSE status.

24-bit Color Depth

For 24-bit color depth[1], the pixel color is stored as three bytes, a red value byte (0-255), a green value byte (0-255), and a blue value byte (0-255). The color values for each pixel are stored in that order. Thus, the color for each pixel is 3 bytes (or 24-bits). So, for a 500 pixel row, there are 1500 (500*3) bytes

The main sets a row value maximum as 15,000 pixels (or 45,000 bytes). The main will perform this verification and display an appropriate error message as required.

1 For more information, refer to: https://en.wikipedia.org/wiki/Color_depth

BMP File Format[2]

The BMP format supports a wide range of different types, including possible compression and 16, 24, and 32 bit color depths. For our purposes, we will only support uncompressed, 24-bit color BMP files. Under these restrictions, the header (or initial information) in the file is a 54 byte block containing a series of data items as follows:

Size	Description
2 bytes	Signature. Must be "BM". *Note*, must ensure is "BM" for this assignment.
4 bytes	File size (in bytes).
4 bytes	Reserved.
4 bytes	Size of header. Varies based on compression and color depth. For an uncompressed, 24-bit color depth, the header is 54 bytes.
4 bytes	Offset to start of image data in bytes. May vary based on compression and color depth.
4 bytes	Image width (in pixels).
4 bytes	Image height (in pixels).
2 bytes	Number of planes in image (typically 1).
2 bytes	Number of bits per pixel. Typically 16, 24, or 32. *Note*, must ensure is 24 for this assignment.
4 bytes	Compression Type (0=none). *Note*, must ensure is 0 for this assignment.
4 bytes	Size of image in bytes (may vary based on compression types).
16 bytes	Miscellanious (not used for uncompressed, 24-bit color depth).

Since the remaining parts of the program will only work for uncompressed, 24-bit color depth, these must be verified before the program can perform image manipulations. Specifically, the following items should be verified:

- File signature is valid (must be "BM" for signature).
- Color depth is 24-bits.
- Image data is not compressed (i.e., compression type must be 0).
- File bitmap block size consistency (file size = size of image in bytes + header size).

Appropriate error message strings are provided in the functions template. Once these are verified, the header information should be written to the output file.

Image Manipulation Algorithms

The following provides the formulas for the image manipulations. To convert a color image to grayscale (black-and-white). Each pixel, (*red*, *green*, *blue*) should set based on the following formula:

$$newRed = newGreen = newBlue = \frac{oldRed + oldGreen + oldBlue}{3}$$

2 For more information, refer to: http://en.wikipedia.org/wiki/BMP_file_format

To brighten or darken the image, for pixel (red, green, and blue) can be computed as follows:

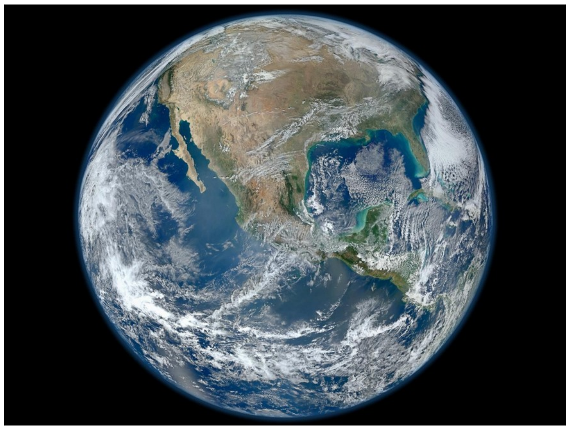
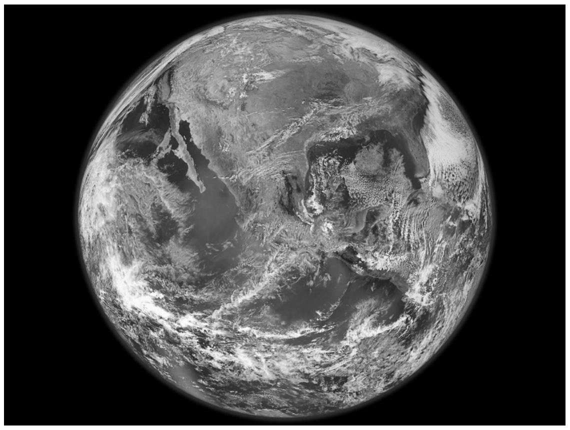
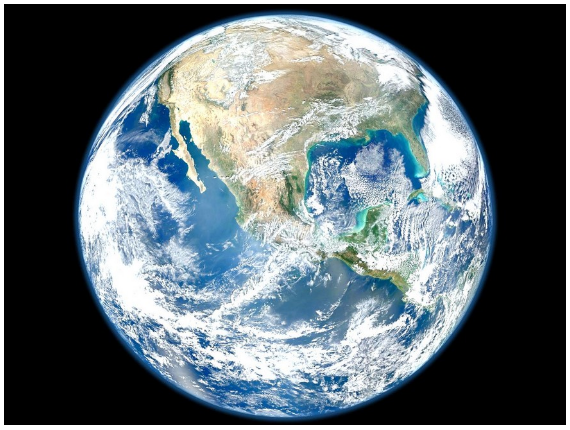
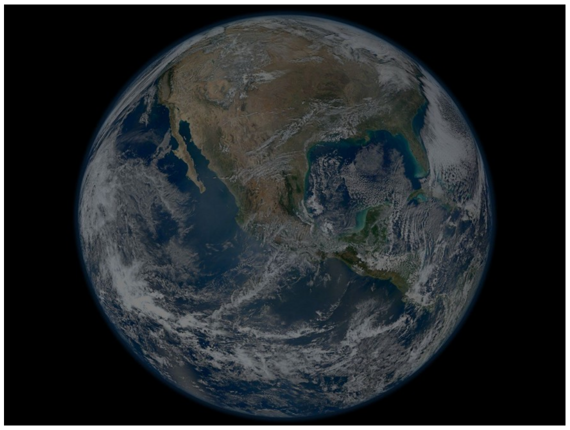
$$newBrightenedColorValue = \frac{oldColorValue}{2} + oldColorValue$$

$$newDarkendedValue = \frac{oldColorValue}{2}$$

Note, pay close attention to the data types. Specifically, the calculations for the brighten operation must be performed as words. If the calculations for the brighten operation exceed the maximum value (255), the new value should be set to 255.

<u>Example Ouptut Images</u>

The following table provides some examples of the expected final output.

Example Output		
Original Image (**`image0.bmp`**)		Output Image converted to grayscale (**`tmpGRimage0.bmp`**)
		
Output Image brightened (**`tmpBRimage0.bmp`**)		Output Image converted to grayscale (**`tmpDKimage0.bmp`**)
		

Example Executions:
The following executions demonstrate a series of errors.

```
ed-vm%
ed-vm% ./imageCvt -grr image0.bmp tmp.bmp
Error, invalid image processing option.
ed-vm%
ed-vm% ./imageCvt -br noSuchFile.bmp tmp.bmp
Error, unable to open input file.
ed-vm%
ed-vm% ./imageCvt -gr image0.bp tmp.bmp
Error, invalid source file name.  Must be '.bmp' file.
ed-vm%
ed-vm% ./imageCvt -gr none.bmp tmp.bmp
Error, opening input file.
ed-vm%
ed-vm%  ./imageCvt -gr image0.bmp tmp.bmpp
Error, invalid output file name.  Must be '.bmp' file.
ed-vm%
ed-vm% ./imageCvt -gr image0.bmp tmp.bmp
ed-vm%
ed-vm% ./imageCvt -dk imageB0.bmp tmp.bmp
Error, invalid file signature.
ed-vm%
ed-vm% ./imageCvt -br imageB1.bmp tmp.bmp
Error, bitmap block size inconsistent.
ed-vm%
ed-vm% ./imageCvt -br imageB2.bmp tmp.bmp
Error, unsupported color depth.  Must be 24-bit color.
ed-vm%
ed-vm% ./imageCvt -br imageB3.bmp tmp.bmp
Error, only non-compressed images are supported.
ed-vm%
ed-vm%
ed-vm%
```

Note, a set of image files, including **`image0.bmp`** (from the example), are provided. You may use BMP images from the net, however, the program will support on any uncompressed, 24-bit BMP format file. You can use the *imagemagik* image processing utility (which will need to be installed).

Submission:

- All source files must assemble and execute on Ubuntu with **`yasm`**.
- Submit source files
    - Submit a copy of the program source file via the on-line submission.
    - Note, only the functions file (**`a11procs.asm`**) will be submitted.
- Once you submit, the system will score the project and provide feedback.
    - If you do not get full score, you can (and should) correct and resubmit.
    - You can re-submit an unlimited number of times before the due date/time (at a maximum rate of 5 submissions per hour).
- Late submissions will be accepted for a period of 24 hours after the due date/time for any given assignment. Late submissions will be subject to a ~2% reduction in points per an hour late. If you submit 1 minute - 1 hour late -2%, 1-2 hours late -4%, … , 23-24 hours late -50%. This means after 24 hours late submissions will receive an automatic 0.

Program Header Block

All source files must include your name, section number, assignment, NSHE number, and program description. The required format is as follows:

```
;  Name: <your name>
;  NSHE_ID: <your id>
;  Section: <4-digit-section>
;  Assignment: <assignment number>
;  Description: <short description of program goes here>
```

Failure to include your name in this format will result in a loss of up to 3%.

Scoring Rubric

Scoring will include functionality, code quality, and documentation. Below is a summary of the scoring rubric for this assignment.

Criteria	Weight	Summary
Assemble	-	Failure to assemble will result in a score of 0.
Buffered I/O	-	Failure to not implment the buffered I/I approach will result in a score of 0.
Program Header	3%	Must include header block in the required format (see above).
General Comments	7%	Must include an appropriate level of program documentation.
Program Functionality (and on-time)	90%	Program must meet the functional requirements as outlined in the assignment. Must be submitted on time for full score.