CSC 250 Program 1
Image Manipulation
Assigned September 19 Due October 9, 2015

Description:

This programming assignment is to help you get a firm understanding of 2d arrays, dynamic memory allocation, ascii data files and binary data files. To accomplish this, we will be using netpbm images. Only 4 of the 6 types will be used and they will have a .ppm or a .pgm extension. Data files with a .ppm extension are color images and files with a .pgm extension are grayscale.

PPM Image Types.

These files are the only ones that we will use for input. The PPM files come with both ascii and binary data for the image. For a file to be a ppm image it must have a magic number of P3 or P6. The magic number is found by reading the first two bytes of the file. Those bytes must match a P3 or a P6 and are case sensitive. The next line can be a comment. If it starts with a # symbol the line is considered to be a comment. Multiple lines can be comments so line 3 could be a comment if the line starts with the symbol #. For our purpose, you will only have one comment line if one exists. After the comment line come the width and height of the image, in that order. A whitespace separates these two values. The next line contains the maximum value that can be in a pixel. For the images that we use, this value will always be 255 or less. If it is greater than 255, you would be dealing with 16 bit pixels; we will only deal with 8 bit pixels. So far all the data is in ascii format. What follows the maximum number is a **single white space** (endl will not work in binary mode) and then the data that makes up the image. If the magic number was P3, the data will be in ascii format.

If the magic number was P6, the data will be in binary format. The image data is stored in the same order for both types of images. The data is supplied for each row. Row 1 being first, followed by row 2, row 3, …. Row n. When working with the data, column 1s data will be first followed by column 2, column 3, ….. Column n. Each column in every row has 3 values, a red channel, a green channel, and a blue channel. They are supplied in that order.
An example ppm file for ascii data for an image with a width of 3 and a height of 2.

Sample display of a P3 image file

```
P3
# Sample data with image columns being 3 and rows being 2.
3 2
255
1
2
3
4 5 6
7 8 9
10 11 12
13 14 15 16 17 18
```

Red channels data is 1 4 7 10 13 and 16
Green channels data is 2 5 8 11 14 and 17
Blue channels data is 3 6 9 12 15 and 18

Viewing the image data in 3 2d arrays (red, green, blue)

Red

1	4	7
10	13	16

Green

2	5	8
11	14	15

Blue

3	6	9
12	15	18

If the data was in binary format, you would not be able to make out the values, but the organization and order of the data would be the same.

Sample display of a P6 image file

```
P6
# Sample data with image columns being 3 and rows being 2.
3 2
255
*&#{|_di^@q~`87+-{
```

PGM Images

PGM image file is a grayscale image file and has magic numbers of P2 and P5. P2 has the image data in ascii format and P5 has the image data in binary format. File format is the same for a pgm that a ppm has up to the image data. When you hit the image data you only have 1 byte for the grayscale value instead of 3 bytes (rgb). You have the possibility of writing an image file out in grayscale. These files will not be used for input.

Sample display of a P2 image file

```
P2
# sample data with image columns being 3 and rows being 2.
3 2
255
1 2 3 4 5
6
```

Grayscale 2d array

1	2	3
4	5	6

Sample display of a P5 image file

```
P2
# sample data with image columns being 3 and rows being 2.
3 2
255
(#*$}[
```

Execution of your program:

I will run your program from the command line and you will have to trap errors such as too many arguments or not enough. Some options will be guaranteed. A minimal run of your program must have 3 arguments supplied to your program.

C:\ prog1.exe -o[ab] basename image.ppm

This is an example where no options have been supplied to change the image. This is useful just to duplicate and image and specified how the data will be formatted.

The first argument in this example specifies how the final image will be stored. If a –oa is specified, the data is to be in ascii format. If a –ob is supplied, the data is to be in binary format. The basename is just the new name for the file. Since no options have been supplied, you will be outputting a color image and you need to add the extension of .ppm to the base name. If a –oa is specified, you will follow the specifications for outputting a P3 image. If a –ob is specified, you will follow the specifications for outputting P6 image in binary format. Remember to change the Magic number according to how it is being changed.

The user could specify and option to be applied to the original image prior to output. Only one option can be specified at a time to keep things simple. The options are specified between the program name and the –o[ab]. Only the following options are allowed.

C:\>prog1.exe [option] -o[ab] basename image.ppm

Option Code	Option Name
-n	Negate
-b #	Brighten
-p	Sharpen
-s	Smooth
-g	Grayscale
-c	Contrast

Notice the # for the –b option. If the –b option is specified, the # is guaranteed to be an integer and it will be supplied.

Negate:

If a (–n) is specified you need to negate each pixel in the rgb arrays. Walk through all 3 dimensional images applying the following equation to each pixel in all 3 color bands.

```
colorband[i][j] =  255 – colorband[i][j]
```

Brighten:

If a –b is specified you are guaranteed that the following command line argument will be an integer between the range of -255 to 255. You need to apply the following formula to every data within the rgb arrays. Process the red, green and blue bands separately. The number is guaranteed to be a valid integer and in the appropriate range.

```
colorband[i][j] = colorband[i][j] + value
```

Sharpen:

if a (-p) is specified you will sharpen the 3 color bands of the image. this is done by subtracting the neighbors form the center pixel multiplied by 5. As you compute each new value, store it into a separate array. This is very important. When done processing a color band, the new array will replace the oldarray.

Newarray

	I,J	

oldarray

a	b	c
d	I,J	f
g	h	i

Assume red band

```
Newred[i][j] = 5 * red[i][j] – b – d – f – h
```

Smooth:

If a(–s) is specified you need to smooth three color bands of the image. This is done by averaging the 9 pixels in a 3 x 3 neighborhood. This must be done to all three color bands. As you compute each new value, store it into a separate array. This is very important. When done processing a color band, the new array will replace the oldarray.

Newarray

	I,J	

oldarray

a	b	c
d	I,J	f
g	h	i

Assume red band

```
Newred[i][j] = (a + b + c + d + red[i][j] + f + g + h + i ) / 9.0
```

The next two options produce grayscale images. When you output the new image, you will need to add a .pgm extension to the basename and follow the output format for a P2 and P5 image type.

Grayscale:

If the grayscale option is specified, you will need a new array the same size as the rgb arrays to hold the grayscale values. Apply the following equation to all pixels to produce a grayscale image.

```
gray[i][j] = .3 * red[i][j] + .6 * green[i][j] + .1 * blue[i][j]
```

You now need to use the magic numbers P2 and P5 when outputting the image and change the extension to a pgm. Only output the gray array.

Contrast:

If the contrast option is specified, you will first need to convert the image from color to grayscale using the procedure described above. While converting to grayscale, keep track of the min and max values.

Next, compute the scale factor using the following equation:

```
Scale = 255.0 / (max – min)
```

Then traverse your 2d grayscale image applying the following equation

```
gray[i][j] = scale * ( gray[i][j] – min )
```

Now output the gray array as a P2 or P5 image.

IMPORTANT NOTE:

No matter what option you are doing, you must maintain the ranges of 0 to 255. If a value exceeds 255 crop it to 255. If a value becomes negative, crop it to zero. You also need to handle rounding when necessary. Your images will be stored in a 2d array of unsigned characters so be careful not to overflow your values and loose information. Consider smoothing. (a + b + c + d + e + f + g + h + i), say a has the value of 250 and b the value of 50. Adding a + b produces a value of 45. Typecast a value in your equation to promote one to a unsigned long and avoid this issue.

Your program will use multiple files. You will create a project that called prog1. Then add in three files. The first file will be named prog1.cpp and will contain your main function. The next two files that you will add into your project will be called function.h & function.cpp. The function.h will contain all of the user defined types, prototypes and includes. Functions.cpp will contain the actual functions the prototypes defined.

Prog1.cpp file will contain the following include

```
#include "function.h"
```

Function.cpp will contain the following include

```
#include "function.h"
```

Function.h will contain the following plus other stuff that you add

```
#include <iostream>
#include <fstream>
:
:
using namespace std;

#ifndef  __FUNCTION__H__
#define __FUNCTION__H__
typedef unsigned char pixel;

struct image
{
    int rows;
    int cols;
    pixel **redgray;     // handles red channel or grayscale
    pixel **green;
    pixel **blue;
};

// place your function prototypes here

#endif
```

You must use the above structure to hold the image information. You may add other fields to the structure if you wish but may not change the existing fields. Pass this structure around to other functions by reference.

Some sample program executions:

Prog1.exe -oa honey bees.ppm
Read in the color image bees.ppm and output it to a file name honey.ppm with the image data in ascii format. Note that the magic number should now be a P3 no matter what the magic number of bees.ppm is.

Prog1.exe -ob honey bees.ppm
Read in the color image bees.ppm and output it to a file name honey.ppm with the image data in binary format. Note that the magic number should now be a P6 no matter what the magic number of bees.ppm is.

Prog1.exe -g -ob bird humming.ppm
Would read in the color image humming.ppm, convert it to grayscale and output the new image to a file called bird.pgm. Its magic number would be P5 for binary.

Prog1.exe -b 90 -ob bird_b humming.ppm
Would read in the color image humming.ppm, add 90 to every pixel in the 3 2d arrays and output the new color image to a file named bird_b.ppm. Its magic number would be P6 for binary.

Prog1.exe -b -20 -oa bird_b1 humming.ppm
Would read in the color image humming.ppm, add -20 to every pixel in the 3 2d arrays and output the new color image to a file named bird_b.ppm. Its magic number would be P3 for ascii.

Prog1.exe -p -ob file balloon.ppm
Would sharpen the image balloon.ppm by using the procedure described above and output the new data to a file called file.ppm. Its magic number would be P6 for binary.

Checking for errors:

Incorrect number of command line options
    Print usage statement and exit
Invalid command line options
    Print usage statement and exit
Invalid magic numbers in PPM files
    Print message, clean up and exit
Dynamic memory fails to allocate
    Print error message, clean up and exit
Files don't open.
    Print error message, clean up and exit

Algorithm:

Check command line options
Get width, height and magic number
Dynamically allocate 3 2d arrays (red, green blue)
Read in image data.
Apply an option if it was specified. You may have to allocate more memory
Formulate new magic number and file name.
Output data into new file in specified format (ascii or binary)
Clean up (free all memory whenever you are exiting and close files)

TimeLine:

Friday September 23	Check for proper usage, opening files and extracting magic number, width and height.
Monday September 28	Dynamically allocate all three color band arrays (red, green, blue) and fill with data from file.
Friday October 2	Write image files out in binary and ascii form with new file name. Do grayscale, negate, and brighten operations. (grayscale must have different format and magic number)
Monday October 7	Do smooth, sharpen, and contrast operations. (contrast is converted to grayscale and must have different format and magic number)
Friday October 9	Program is due. Submit your three files within a zip named prog1.zip. Do no use any other form of file (i.e. rar, tgz, bz …) Your program must be documented to use doxygen.

Helpful Hits:

Don't panic, work on small sections at a time and don't be afraid to write functions.
Save and compile often.
Work on the project according to the time line.
Don't wait until the last minute:
If you have problems, try to figure it out, but don't be afraid to get help
Work on your program frequently, trying to do large pieces of your program at one time often leads to careless mistakes.

Generating and viewing your images:

Go to http://www.gimp.org and download binary package for windows and install.
Open up any image that you wish and select save as and change the extension to ppm. When you save the image it will ask you for which format.

You may view your work in gimp. Adobe paintshop supported netpbm.