Working with PPM images, for the most part, are fun projects. As we go through this semester we are going to use this fun concept to learn and practice various computing concepts.

**ASSIGNMENT OVERVIEW:**

Often astronomers, or individuals interested in astronomy, will take a varying number of pictures of an object in outer space such as, stars, planets, galaxies, comets, etc. NASA’s Hubble Site has a ton of images taken by the Hubble Space Telescope. <http://hubblesite.org/gallery/> However, often these images may be grainy or have random noise in the images. To remove this noise, the photographer or in this case the Hubble Space Telescope will take several images then process the images using a technique called image stacking. Another problem a photographer (professional or not) sometime encounters is the photograph may have some unintended object in the picture. With the luxury of digital cameras, we have the option of taking several snapshots of the same image, then applying a filter to the images to remove the unwanted objects (bird, ball, etc.) For this assignment, you are going to write the code necessary to solve the two problems discussed above. As mentioned in class, you are also going to write a function that will check for comments and whitespace when reading the image header. There are other functions you will need to implement, most of which are described below in the requirements section.

**LEARNING OBJECTIVES:**

This assignment will give you practice working with the following concepts:

* Multiple files
* Command-line arguments
* Pointers
* Array of pointers
* Structs
* Parsing files
* C – I/O concepts (fprintf, sprintf, FILE\*, etc.)
* Dynamically allocating memory
* Sorting Algorithms
* Reading and writing images
* FILE I/O
* Many other concepts

**ACADEMIC INTEGRITY:**

This is an individual assignment. You may not receive help from anyone other than myself a lab TA or ASC tutor. Please review the academic integrity policy provided in the syllabus.

**REQUIREMENTS:**

I will provide a new ppm\_utils.h file that includes the functions you will need to implement. Outlined below are several functions you **MUST** write.

1. **void skip\_ws\_comments(FILE\* infileptr);** This function will be used when you are reading the header of the ppm files. As discussed in class, **ppm** headers can and often do have comments embedded within them. A header will not have a comment before the “Magic Number” -- P6 or P3. A comment can be on the same line as the Magic Number. As an example:

P6 #Hello world

A comment will always end with a new line characters. Therefore, you will not have something like the following:

200 200 #Hello Again 255

Comments can be on a line by itself.

#hello wold

Here is an example header with comments and spaces:

**P3 #Hello World, this is a comment**

**#Next will be a set of blank spaces and then the width**

**200 #And another comment**

**200 255**

Your function should ignore these comments and extra white spaces.

Here are the suggested steps needed for this function.

* Use the c function **fgetc** to read a character.
* Check to see if the character read in step 1 is a # or space.
  + You can test for a space using the “C” function **isspace**. If it is a space then continue reading, using **fgetc**, until it is no longer a space. Once you determine the character is no longer a space you need to move the pointer back to the previous character (remember we discussed in class what happens when you read something from a file using a file pointer) You can move the pointer back by using the function **ungetc.** Ungetc is the equivalent of putting the character back so it can be read again during the next read cycle.
  + If the character read in step 1 is a # then continue reading until you read a ‘\n’ character.

This function should have one or more looping mechanisms.

I strongly suggest you write a test driver (main) that will create a small ppm image that writes comments in the header. You can then use that image to test your skip\_ws\_comments function.

Once you have written **skip\_ws\_comments**. You will need to modify the read\_header file to do the following:

* Read the magic number
* Check for comments and WS
* Read the width
* Check for comments and WS
* Read the height
* Check for comments and WS
* Read the Max Color Val
* Check comments

1. **void openInputFiles(char\* name, FILE\* input[]);** The first parameter represents a word passed to main through command line arguments. The second parameter represents an array of FILE pointers.

This function will create and then open, for reading, the input files. You will use the function **sprintf** to create a set of ppm file names. Each of these files should be opened for reading and stored in the array of file pointers passed to the function. You **can not** hard code these file names. You **must** generate the file names using sprintf. The file names will be in the form of <name>\_001.ppm. If the parameter (name) is “average” then you will create and open, for reading, 10 files. If the parameter (name) is “median” you will create and open, for reading, 9 files. I will provide you with two sets of ppm files. One set has the name of average\_001.ppm, average\_002.ppm through average\_010.ppm. The other set has the name of median\_001.ppm, median\_002.ppm through median\_009.ppm. When opening the files, you must check that the file open successfully. If it did not open successfully, print an error and return 1.

1. **imate\_t\* removeNoiseAverage(image\_t\* img[]);** The parameter represents an array of image\_t\* that point to dynamically allocated memory for the images used in this function. This function will use the images called average\_001.ppm through average\_010.ppm.

This function will remove the noise or graininess in photos taking by the hubble space telescope. For each pixel calculate an average of the RGB channels of the 10 images read in. You will need to create a local variable of type imate\_t \* to store the pixels for the output image. (Don’t forget to dynamically allocated the memory for the local image\_t\*.) The calculated average should be written to output variable.

Ex. For the first pixel of all 10 images you will read the red values and average them, the green values and average them, and the blue values and average them. Then save the average values in the first pixel of the local image\_t\* created for the output. Do this for all pixels.

1. **image\_t\* removeNoiseMedian(image\_t\* image[]);**The parameter represents an array of image\_t\* that point to dynamically allocated memory for the images used in this function. This function will use the images called median\_001.ppm through median\_009.ppm.

To remove an unwanted object in a series of images you can use a method similar to that used in removeNoiseAverage function. Rather than calculating the average you calculate the median. This will involve, for each pixel, reading the RGB values for all 9 of the images, store these values in an arrays – one array for Red, Green, and Blue. Sort the 9 values in each array. Create a local variable of type image\_t\* for the output. Write to the output the median (middle) RGB values. (Don’t forget to dynamically allocated the memory for the local image\_t\*.)

1. **void sort(unsigned int\* arr, int n);** The function described in 4 above requires you sort the values in the array passed to the function. This is what the sort function will be used for.
2. You are to write a driver called **main.c**; The driver should have two variables of type FILE\* arrays -- one for the pointers to the **average** ppm files and one for the pointers to the **median** ppm files. It should also have two variables of type image\_t\* arrays – one for the pointers to the **average** images and one for the pointers to the **median** images. You should open the appropriate files, read the ppm files, call either removeNoiseAverage or removeNoiseMedian. If removeNoiseAverage was called the output should write to P6. If removeNoiseMedian was called the output should write to P3.

The command to execute the program should be as follows:

**<executable> <string average or median> <output file name>**

**./a.out average output.ppm**

You should close the open files and free any dynamically allocated memory.

Above are the functions you **MUST** implement for this project. Depending on how you write these functions, there may be other functions you need to write for this assignment, such as a swap to use with the sort function.

You may not change the signature of the functions provided in the ppm\_utils.h file. **I plan to test your program with various drivers developed by me.** If you change the functions I will not be able to test the program with my drivers. You will loose points for this.

**FORMATTING:**

You will need to add a header to each of your files like the following:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*Your name

\*CPSC 1020 your Section, Sp18

\*Your email

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Your program should compile with no warnings and no errors. If your program does not compile the highest grade you can get for the assignment will be 20. If your program compiles but has warnings, there will be a deduction up to 20 points.

* Your code should be well documented. (comments)
* There should be no lines of code longer than 80 characters.
* You should use proper and consistent indention.

**HANDIN:**

Use handin.cs.clemson.edu to submit your files. I have created buckets named PA1.

Things to do prior to handing in your files:

1. **Test your program on the SoC servers**. I will not accept the excuse “It compiled on my computer.” I test programming assignments on the SoC servers. To compile the program you should do the following:

**gcc main.c ppm\_utils.c –Wall –o main**

1. Tar zip your files naming the tarred file PA1.tar.gz. Your tar file should **NOT** have nested files.
2. You should provide a **README** that consist of the following.

* A short description of any problems you encountered when writing this program.
* How you solved the problems you encountered.
* Your thoughts on the assignment. This is your opportunity to tell me if you like the assignment or not. What you did or did not like about the assignment. Anything you want to tell me.

Here is an example of an image that has noise in it and the output after the noise has been removed. Basically, I took an image from NASA’s website, ran it through a program that introduced random noise in the image. Then ran it through the removeNoiseAverage function and the one on the right is the output.



This is the original image from NASA.



Here is a sampling of the images used to remove the unwanted object in the image (the person).

  



Here is the result after calling the removeNoiseMedian function with these images:

