# Lab 10 – C++ Classes and PPM Images

CPSC 1021- Fall 2019

## Due date: Sunday,November 17 @midnight

## Introduction

You were introduced to working with PPM files in lab 4. Lab 4 used 2 structs; 1 to represent the header of a ppm and 1 for the pixels of a ppm image.

In today's lab, you are going to implement two classes, Header and Pixel both of which will represent ppm images. You are then going to write 4 driver files. Each of which will read in a ppm file, flip it horizontally, and write it back out. Each driver will store the pixels using different methods (1D Array, 2D Array, 1D Vector and a 2D Vector.)

## Lab Objectives

- Practice with classes
- Image processing and manipulation
- Using vectors
- Using 2D vectors
- Working with dynamically allocated memory (1D and 2D)

## Lab Instructions

I have provided you with header.hpp and pixel.hpp. You will implement the functions provided in each .hpp file. While I realize you may or may not use all of the functions, for practice you are still required to implement all of the functions provided in the .hpp files. You will then create 4 driver files named vec1DDriver.cpp, vec2DDriver.cpp, array1DDriver.cpp, and array2DDriver.cpp. As you probably have discerned, each of the driver files will use a different storage method for the image you will read.

What each driver should do:
Basically, each driver will read in a ppm image, flip the image horizontally and write it to the output file.

Create the input and output file stream. Each will be provided through command line arguments.

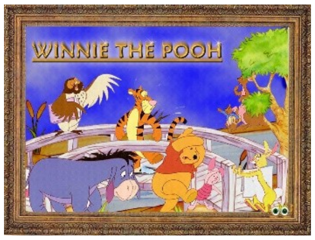
Allocate the space for the image. You can determine the amount of space needed from the information you read from the header. Read the pixels from the input image, flip the image horizontally and write them back out.

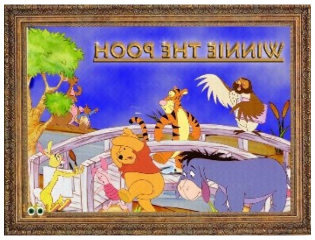
The horizontal flip function is not part of either class it should be implemented as an independent function in functions.h and functions.cpp. Below is the prototype for the horizontal flip function for the array2DDriver.cpp file. The prototype will change for each of the 4 driver.cpp files.

```
void HFlip(Header, Pixel**);
```

The **HFlip** function will iterate through all pixels providing the necessary code to flip an image horizontally. As an example, suppose I have a 3X3 image. The **Original** matrix below represents the image with the pixels numbered 1-9. Upon completion of the **HFlip** function the pixels should be in the order displayed by the **Horizontally Flipped** matrix**.**

| Original | | | | Horizontally Flipped | | |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | | 3 | 2 | 1 |
| 4 | 5 | 6 | | 6 | 5 | 4 |
| 7 | 8 | 9 | | 9 | 8 | 7 |





You are also required to make sure you check that all input files open successfully. If they do not open successfully you should print a message and exit the program. You should check that the appropriate number of command line arguments are passed to the command line. If not print a message and exit the program. Rather than checking these in the main, you are required to create a function for each of these tasks. These functions are to be implemented in the functions.cpp and the prototypes should be placed in the functions.hpp file.

Additional information:

**Vector of vectors (multidimensional):**

There are several examples of how to create multidimensional vectors online. I have given you several links to look at:

https://www.geeksforgeeks.org/2d-vector-in-cpp-with-user-defined-size/
https://stackoverflow.com/questions/823562/multi-dimensional-vector
https://www.codeproject.com/Questions/456547/How-to-use-Multidimensional-vector-in-Cplusplus

Here is an example of a 2D vector I used to practiced with.

vector< vector<int> > test;

This is an empty vector of vectors of type **int** called **test.**

I used resize to create memory for the 2D vector:

Since you know the exact size of  the 2D vector using resize is more efficient than using Push_back().

test.resize(10, vector<int>(5));

I now have a 2D vector called test that is a 10 X 5 vector (10 rows, 5 cols). You can access the vectors using array notation or by using the safer function **at()**. I could have initialized each of the elements to a value when calling resize.

## What to turn in

- **header.cpp, header.hpp, pixel.cpp, pixel.hpp, functions.hpp, functions.cpp, vec1DDriver.cpp, vec2DDriver.cpp, array1DDriver.cpp, array2DDriver.cpp**

## Compile and Execute

Use g++ to compile your code as follows **and include the c++11 standard!**:

*g++ -std=c++11 -Wall -g pixel.cpp header.cpp vec1DDriver.cpp -o vec1DDriver*

*or*

*g++ -std=c++11 -Wall -g pixel.cpp header.cpp vec2DDriver.cpp -o vec2DDriver*

*or*

*g++ -std=c++11 -Wall -g pixel.cpp header.cpp array1DDriver.cpp -o array1DDriver*

*or*

*g++ -std=c++11 -Wall -g pixel.cpp header.cpp array2DDriver.cpp -o array2DDriver*

Execute the programs

*./vec1DDriver pooh.ppm vec1DOut*

*./vec2DDriver pooh.ppm vec2DOut*

*./array1DDriver pooh.ppm array1DOut*

*./array2DDriver pooh.ppm array2DOut*

```
// Sample Header
/*******************
  your name
  username
  Lab 1
  Lab Section:
  Name of TA
*******************/
```

FORMATTING:

1. Your program should be well documented
2. Each file should include a header:
3. Your program should consist of proper and consistent indention
4. No lines of code should be more than 80 characters

5 – 10 points will be deducted for each of the above formatting infractions.

Submission Instructions

- Test your program on the School of Computing server prior to submitting.
- Use the tar utility to tar.gz all source files. **Do not tar an entire directory! When I untar your archive, I should see all the files you included, not a top-level directory! Failure to correctly tar may result in up to a 25-point penalty!**

- Name your tarred file **<username>-lab<#>.tar.gz** (ex. yfeaste-lab10.tar.gz)
- Use handin (http://handin.cs.clemson.edu) to submit your archive