CS515 Assignment 3

The purpose of this assignment is to practice working with multidimensional arrays and file input/output. Download all starter files from `~cs515/public/3P`

In this assignment, you will work on simple image manipulations. An image is a matrix of pixels represented by a two-dimensional (rectangle) array. The dimensions (width and height) of the array is called the resolution of the image. There will be **`m`** x **`n`** pixels in an image, where **`m`** and **`n`** are the width and height of the image. E.g., an image of resolution 320 x 240 has 76800 pixels; the width of the image is 320 and height 240.

The PGM format is a gray-scale image format. It is designed to be extremely easy to learn and write programs for. PGM format images typically have a file name extension **`.pgm`**. It has two types, containing magic identifiers P2 and P5. PGM image in P2 format stores pixel values as the ASCII characters for the digits, delimited by spaces between consecutive pixel values. PGM image in P5 format writes the bytes of binary numbers without delimiters. In this assignment, we will use the P2 format.

Each PGM P2 format image consists of the following:

- A "magic number" for identifying the file type. An ASCII PGM file's magic number are the two characters "P2".
- Whitespaces (newline, space, TAB).
- A width, formatted as ASCII characters in decimal.
- Whitespaces.
- A height, again in ASCII decimal.
- Whitespaces.
- The maximum gray value, again in ASCII decimal. (Typically 255)
- Whitespaces.
- An array of gray values (in ASCII decimal) between 0 and the specified maximum value, separated by whitespace, starting at the top-left corner, proceeding in normal English-reading (row-major) order. A value of 0 means black, and the maximum value means white.

The following is a 24x7-pixel PGM P2 image file containing a maximum gray scale value of 15:

```
P2
24 7
15
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  3  3  3  3  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0 15  0
0  3  3  3  0  0  0  7  7  7  0  0  0 11 11 11  0  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0  0  0
0  3  0  0  0  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

How can you display a PGM image?

PGM images can be displayed by many image viewers (e.g. `xv` on Unix, GNU GIMP, ToyViewer on mac). If you don't have one installed on your computer, you can convert the PGM image into other format. E.g., you may convert it into a PDF file or a PNG image (viewable in a web browser). On agate, you may use **`/usr/bin/geeqie`** program to display the PGM image directly. Or, you may use the **`convert`** command to create PNG and PDF format given the original PGM image.

```
%convert pepper.pgm pepper.png
%convert pepper.pgm pepper.pdf
```

Part 1 (50%)

Write a program to read an image of PGM P2 format, and produce its negative image and its 90-degree clockwise rotation. Below is an example of an image (`pepper.pgm`) with its two output variations.

Original image

Negative

90-degree clockwise rotation

- Your program should take three command-line arguments as shown below:
  `image <image_input_file> <negative_image_filename> <rotate_image_filename>`

  The program **`image`** reads an existing PGM image, and outputs the image's two variations to files as given at command line (respectively). You need to provide error checking for command line arguments, and an error in the number of arguments should supersede any errors from files that cannot be found. You may assume the input file is a PGM P2 image in proper format without comments (lines starting with "#"), and the maximum grey value is 255.

  Here is a sample run:

```
% ./image
Usage: image <input image> <neg image> <rotate image>
% ./image pepper.pgm
Usage: image <input image> <neg image> <rotate image>
% ./image xxx.pgm n.pgm f.pgm
Can not open xxx.pgm for input.
% ./image pepper.pgm pn.pgm pr.pgm
%
```

  You must use dynamic two-dimensional arrays for storing the images. You need to properly allocate and de-allocate the array so the program does not contain any memory leaks. Valgrind checks should reveal no errors.

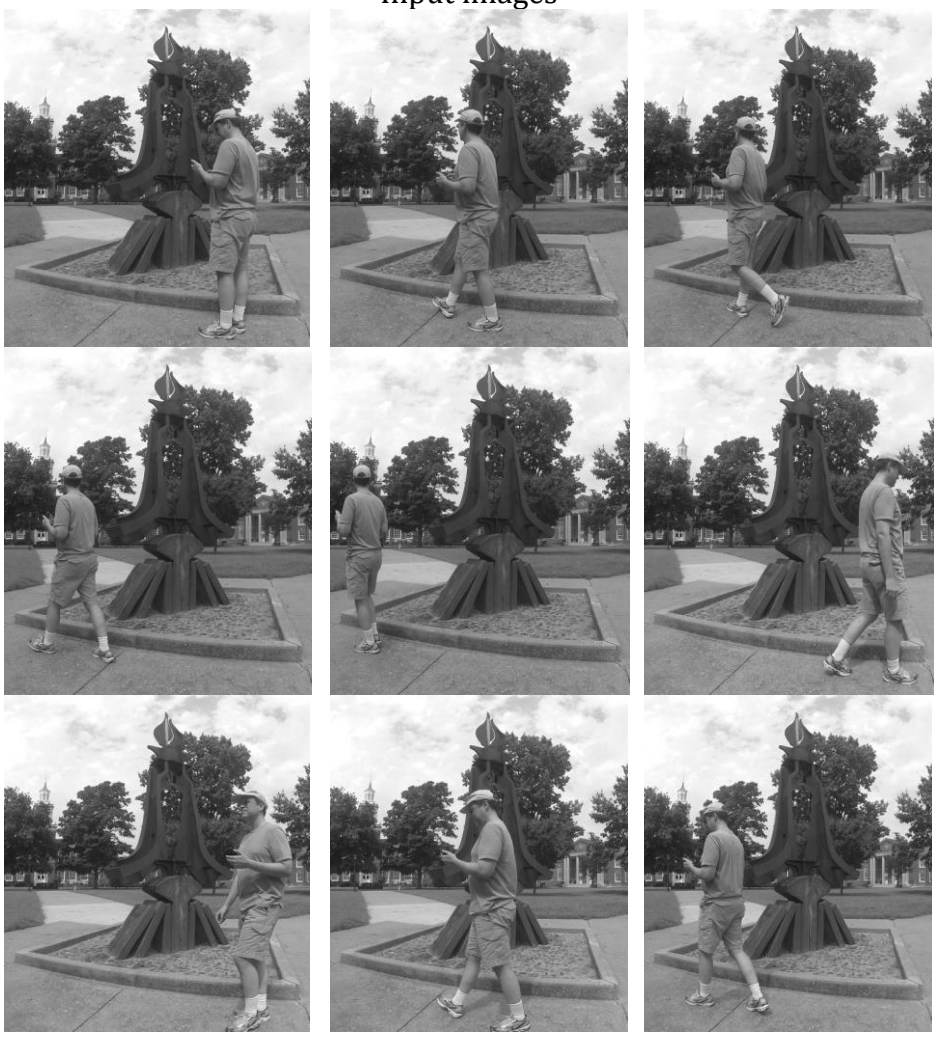
Part 2 [1] (50%)

You are given 9 pgm images, all showing a nice sculpture… but also a curious visitor who happened to be around when the images were taken. Write a program that can "filter" out the visitor and produce a nice clean image of the structure and background only, without the person.

In the images on the next page, you can see that most of sculpture is visible in each of the 9 images and the person only blocked part of the sculpture. Since he is in a different position in each photo, he blocks a different part of the sculpture each time. Your job is to determine which pixel grey levels are part of the sculpture and the background (what you want), and which pixel colors are part of the person (what you don't want). Then you could create a photo without the person.

[1] adopted from Stanford nifty assignment 2014 collection http://nifty.stanford.edu/2014/nicholson-the-pesky-tourist/student.html

Input images



Expected output image

We assume that most of the pixels in each image at a specific coordinate (x, y) have the same grey level value, or are very close in value. These are the values we want in the final image at that coordinate. However, sometimes a particular pixel in a particular image will be part of the person, and that pixel will have a very different value from the other pixels at the same coordinate in the other images. These "outliers" are the noise that you don't want in the final image. Use an image processing technique that finds the median value of the pixel across similar images (image stack) to remove the noise and reveal the "average" image.

Name your program imagestack.cpp. On the command line, it should accept up to 9 input image filenames, followed by the name of a file for storing the final filtered output image, like this:
`./imagestack <image1>.pgm <image2>.pgm … <last_image>.pgm <output_file>.pgm`

A sample run:
`./imagestack pt0.pgm pt1.pgm pt2.pgm pt3.pgm pt4.pgm pt5.pgm pt6.pgm pt7.pgm pt8.pgm output.pgm`

You must use a dynamic 3-dimensional array for storing the images. You need to properly allocate and de-allocate the array so the program does not contain any memory leaks. Valgrind checks should reveal no errors.

Submission:

- Submit your source files but not your **`makefile`**.
- You must also fill out the README file and submit it as well. Submit the following files to the appropriate assignment on Mimir:

  `image.cpp imagestack.cpp README`
- Do not turn in executables or object code. Programs that produce compile-time errors or warnings will receive a zero mark (even if it might work perfectly on your home computer).
- Be sure to provide comments in your program. You must include the information as the section of comments below:

```
/**    CS515 Assignment 3
File: XXX.cpp
Name: XXX
Section: X
Date: XXX
Collaboration Declaration: assistance received from TA, PAC etc.
*/
```

Some notes on grading:

- Programs are graded for correctness (output results and code details), following directions and using specified features, documentation and style.
- Here is a tentative grading scheme.

image command line arguments 4 test cases	10
image test file 1 (square: negative/rotate)	10
image test file 2 (rectangular: negative/rotate)	10
image test file 3 (square: negative/rotate)	10
image Valgrind check	10
imagestack test 1	20
imagestack test 2	20
imagestack Valgrind check	10
others	
not using 2D dynamic array in part 1	-80
not using 3D dynamic array in part 2	-80