CS202 -- Lab 2

- CS202 -- Data Structures & Algorithms I
- James S. Plank
- Fall, 2021
- Directory for files, makefile, executables and gradescript: **/home/jplank/cs202/Labs/Lab2**
- This file: **http://web.eecs.utk.edu/~jplank/plank/classes/cs202/Labs/Lab2/**

Executables, Formatting and Vector Functions

As always, there are executables for each part of the lab in the lab directory. In the first part (**pgminfo**) your program's output needs to match mine exactly.

In the rest of the parts, your program needs to match mine exactly when there is an error. If there is not an error, then your program's PGM must be equivalent to mine, but the formatting may be different.

In the places where your program has to match mine exactly, if you have questions about what the output should be, try checking the executable first rather than asking the TA. Often, the answer is there.

For my PGM output files, I often just print a pixel on each line. That simplified my programs.

You may not use the vector function **reverse()**. Part of the goal of this lab is for you to manipulate vectors directly.

Starting Out

- Make your **lab2** directory.
- Make a **bin** and **src** directory.
- Copy **/home/jplank/cs140/Labs/Lab2/makefile**.

I know y'all get confused about what to copy, and your tendency is to copy everything, so here is some text on what to copy:

Whether you copy **gradescript/gradeall** is up to you, because you can call it from the lab directory.Â The **Gradescript-Examples** directory is useless to you, because the gradescript uses it from the lab directory.Â Whether or not you copy the pgm files is up to you.Â It's probably more convenient to do so, but certainly not necessary.Â The jpg files and index.html are also useless.

PGM Files

This lab lets you create and manipulate PGM files. PGM stands for *portable gray-map*, and is file format for non-color digital pictures. PGM files are nice because they are ASCII files that you can view with text editor like **vi**, and that you can also view with certain software.

Unfortunately, most web browsers do not have support for PGM files. However, every environment has many ways of dealing with them. For example, Photoshop, Open Office and Gimp all support them as image types. I would guess that iphoto and whatever photo editor comes with Windows handles them as well.

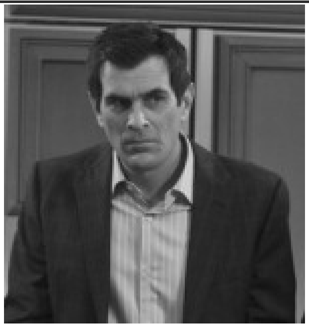
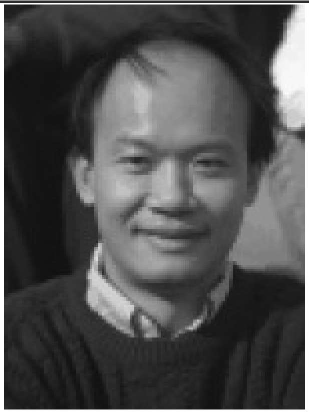
Most Unix environments have a program called **convert** (from https://imagemagick.org), which will convert pgm's to jpgs for easy viewing. For example, on our labs:

```
UNIX> cp /home/jplank/cs140/Labs/Lab2/pgm/Nigel.pgm .
UNIX> convert Nigel.pgm Nigel.jpg
```

You may now view **Nigel.jpg** with a web browser. Test it out. It should look like:

Here are a bunch of sample PGM files that you can use with this lab. They contain some of my most important role models (well, as of around 2007). Unfortunately, you shouldn't try to create your own PGM files with the above software, as most PGM converters convert to a different format, or they include comments, which I'm not requiring you to handle in this lab.

Also, the sizes of the JPG pictures below do not match the PGM files. I've shrunk some of them to make them display better.

pgm/Red.pgm Red Forman - Role model for being a father and husband.	 pgm/Phil.pgm Phil Dunphy - Ditto	pgm/Hand.pgm Mr. Hand - Pedagogical role model
pgm/Schalk.pgm Schalk Burger - Sportsmanship	pgm/Frank.pgm Frank Slade - Dealing with children	 pgm/Kai.pgm Kai Li - Motivational

pgm/Nigel.pgm Nigel Tufnel - Artistic	pgm/Paris.pgm Paris Gellar - People skills	pgm/Pike.pgm Rob Pike - Role model for answering questions during talks
pgm/Reggie-Mike.pgm Reggie and Mike - Parental	pgm/Rodney.pgm Rodney Farva - People skills	pgm/Sheldon.pgm Sheldon Cooper - For what is Sheldon not a role model?

A picture is simply a two-dimensional array of *pixels*. In a PGM file, a pixel is a gray value between 0 and 255. Zero is black, and 255 is white. Everything in between is a different shade of gray.

PGM files have a specific format. (Actually, they are more robust than this -- I've removed comments to make them easier for you). A PGM file is composed of words. The first word is ``**P2**''. Then the next two words define the number of pixels in the picture -- first is the number of columns in the two-dimensional array, and then the number of rows. The next word is the number 255. So, if you look at the beginning of **pgm/Red.pgm**, you'll see:

```
UNIX> head pgm/Red.pgm
P2
235 183
255
 67  74  76  80  76  77  70  67  75  82 102 119 129 133 130 127 128
130 134 131 130 129 130 133 132 125 116 109 101 102 105 105 113 119
128 133 138 137 137 137 141 143 143 144 140 143 146 144 140 138 128
122 118 107 106 104 106 107 115 124 127 121 114 108 103  98  97 108
122 138 140 140 139 129 123 125 127 141 145 147 147 146 147 147 147
149 150 150 150 155 168 185 195 198 202 201 201 200 200 199 201 199
200 201 207 205 206 203 204 205 206 206 205 206 207 207 207 207 207
UNIX>
```

This means that **pgm/Red.pgm** is a 183 * 235 array of pixels. After the 255 come all the pixels. First, the pixels in the top row, then the pixels in the next row, etc. Note that the ASCII formatting of the file doesn't mean anything -- there could be one pixel per line and the file would still be a legal PGM file. In **pgm/Red.pgm** above, the first 235 pixels are those in the top row, then the next 235 are in the second row, and so on. There will be a total of 183*235 = 43005 pixels. After the last pixel, the file ends.

Before you go any further, create a PGM file of your own -- make it 10x10 and give the pixels any value you want. Take a look at it. Something like:

```
P2
10 10
255
0 10 20 30 40 50 60 70 80 90
10 20 30 40 50 60 70 80 90 100
20 30 40 50 60 70 80 90 100 110
30 40 50 60 70 80 90 100 110 120
40 50 60 70 80 90 100 110 120 130
50 60 70 80 90 100 110 120 130 140
60 70 80 90 100 110 120 130 140 150
70 80 90 100 110 120 130 140 150 160
80 90 100 110 120 130 140 150 160 170
90 100 110 120 130 140 150 160 170 180
```

This should look like:

Cool, no?

Program #1: Pgminfo

Your first PGM program should take a PGM file on standard input and report the number of rows, the number of columns, the total number of pixels, and the average value of all the pixels, padded to three decimal places. Your program should work on all valid PGM files, and should print out an error (using **cerr**) on any invalid PGM file. Examples of invalid PGM files are:

- Those that don't begin with P2.
- Those that don't have non-negative integers after the P2.
- Those that don't have the number 255 after the number of rows and columns.
- Those that contain the wrong number of pixels after the P2. This includes having too many pixels.
- Those that contain pixels whose values are not numbers between 0 and 255.

Here is an example of **pgminfo** running on some of the PGM files. Note, I have a few bad PGM files in **pgm/Bad-1.pgm**, etc. You need to make sure that your output matches mine *exactly*. And I mean exactly, meaning the same punctuation, same number of spaces, and same capitalization. Use **printf()**, not **cout**.

```
UNIX> bin/pgminfo < pgmRed.pgm
# Rows:         183
# Columns:      235
# Pixels:     43005
Avg Pixel:  120.142
UNIX> bin/pgminfo < pgmPike.pgm
# Rows:         235
# Columns:      197
# Pixels:     46295
Avg Pixel:   99.932
UNIX> bin/pgminfo < pgmBad-2.pgm
Bad PGM file -- No column specification
UNIX> bin/pgminfo < pgmBad-5.pgm
Bad PGM file -- pixel 99 is not a number between 0 and 255
UNIX> bin/pgminfo < pgmBad-6.pgm
Bad PGM file -- Extra stuff after the pixels
UNIX>
```

When I print a pixel number, it is zero-indexed. So the first pixel is pixel zero.

Program #2: Bigwhite

This program takes two numbers as its command line arguments -- the number of rows and the number of columns. It then writes a PGM file on standard output which contains that number of rows and columns, all of white pixels. Again, you should error check to make sure that the proper number of command line arguments are given, that they are integers and in the proper range. On an error, print the error statement to **stderr**. As an example, try:

```
UNIX> bin/bigwhite 20 10 > a.pgm
```

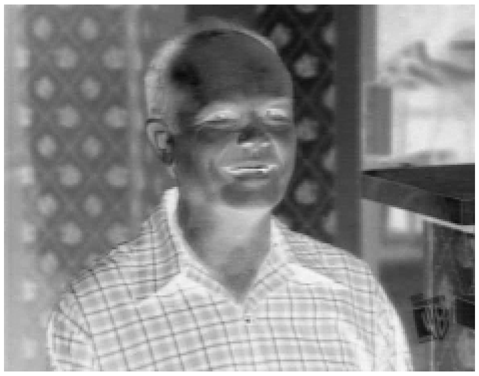
This will create a PGM file **a.pgm**, which has 20 rows and 10 columns of white pixels.

Your output in this and the next three programs should match mine exactly when there is an error. Otherwise, the PGM files that it emits should be equivalent to mine (they should make the same picture), but do not have to have the same output exactly.

Program #3: Neg

Neg takes a PGM file on standard input, and prints a PGM file on standard output that is the negative of the input file. If the PGM file is not valid (same parameters as **pgminfo**), print an error to standard error.

For example, here is the negative of **pgm/Red.pgm**:

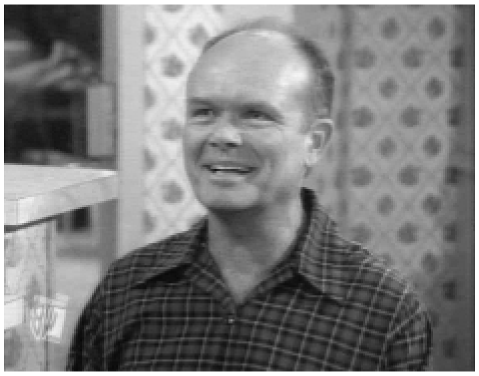


Program #4: Hflip

Hflip reads a PGM file on standard input, and prints a PGM file on standard output. The output file should be the horizontal reflection of the input file -- in other words, left is right and right is left.

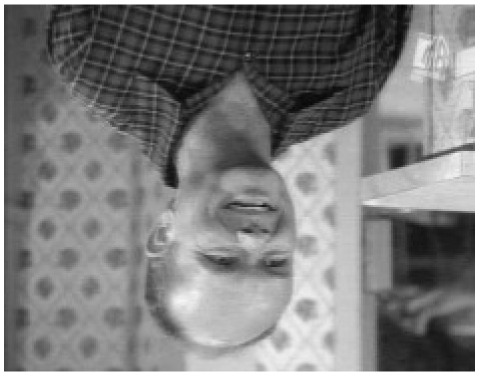
You'll have to use a vector for this program.

Here's the **hflip** of **pgm/Red.pgm**:



Program #5: Vflip

Vflip is just like **hflip**, only it flips the file vertically:



Just a little about the gradescript

For all parts but part one, I use the program **bin/pgm_compare** to compare your output with mine. This program takes two command line arguments, which should each be PGM files, and compares them to make sure that they are equivalent, even if their formatting is different.