README:

Matthew Langton

I have completed this assignment by myself with a bit of help from TAs and the professor through EdStem.

All parts of my program function properly including the Array2, rotation functions for 0, 90, 180, and 270 degrees, flip functions for vertical and horizontal flips, and a transpose function.

Architecture Documentation:

ppmtrans features a command line argument struct which takes in values of --rotate, --flip, and --transpose. --rotate is followed by a value of 0, 90, 180, or 270, and --flip is followed by a value of either vertical or horizontal. All inputs can also be followed by either --row-major or --col-major in order to specify whether to do the desired operation in row major or column major order. These command line arguments will call on functions in main.rs (rotate(), flip(), and transpose()) which will then call on the related functions in array2 which take in an array2 object, perform the desired operation, and return an array2 which has had some operation done to it.

I have spent approximately 10-11 hours completing this assignment including the design document, estimations, coding, and benchmarking work.

Part C:

Table of Running Times For Each Function (Values were averaged over 3 runs):

Elapsed Time Table	row-major	col-major
180 Degree	3.095 seconds	4.2536 seconds
90 Degree	3.6403 seconds	3.797 seconds

These measurements somewhat match what I had expected from my estimates in part B. The two row-major versions of the program run faster than the two column-major versions. This is because as per my justification from part B, the actual degree of the rotation has little effect on cache hit rate with only potentially a small effect based on the size of the image, as the data locality remains the same regardless, so this will not change much for cache hit rate, and the difference between row and column major is much higher because row-major access will result

in a very high cache hit rate as a result of the data locality within rows of a 2D array being better than the data locality within columns. What didn't match my prediction however was the speed of the 180 degree column major operation. I had predicted that this would be faster than the 90 degree column major operation which was definitely incorrect. This being the slowest of the 4 operations was likely because considering the operation is column major and that the image is being rotated 180 degrees, the cache hit rate is very bad as the pixels are being moved very far away from their original locations (i.e. bad data locality).

Part D:

One possible way of designing an Array2 to achieve better cache hit rates with better data locality would be to design an Array2 using a block/tile based layout for storing the data. The way this would work is the Array2 would be divided into 'chunks' of some size, be it 8x8 or 16x16 or some other number which could be chosen based on what works best with the cache. This would result in better cache hit rates as data within the Array2 would be processed based on these 'chunks' resulting in data being processed together causing much better cache hit rates than my current implementation which simply uses one giant Vec to store everything which causes inconsistent cache hit rates as data may or may not be processed with other data within the Vec.