Locality Design
Yoda Ermias (yermia01) and Maiah Islam (mislam07)

Files to make:

- ☐ Uarray2b.h/c
- ☐ A2plain.c
- ☐ Ppmtrans.c/h

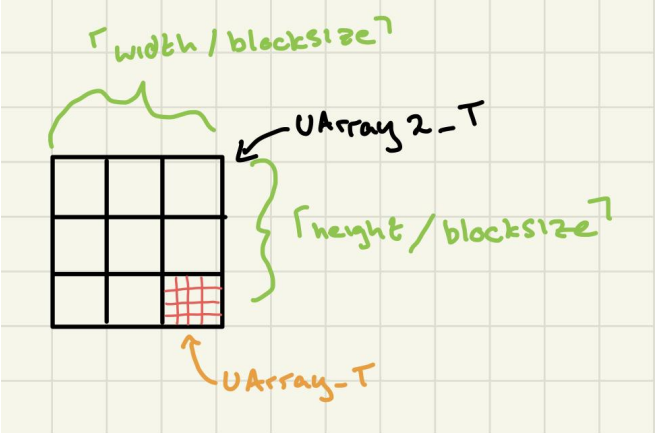
As suggested, we will use a UArray2_T to implement the UArray2b, where each element of the UArray_T contains one block, and one block is a single UArray_T.

Implementation Plan for Part A (Implementation for the Creation of UArray2b):

Structure for UArray2b

-Use a UArray2 as our main structure for the UArray2b

- The width of the UArray2 will be the ceiling of width / blocksize
- The height of the UArray2 will be the ceiling of height / blocksize
- Each index of the UArray2 will hold a UArray of the actual pixels of the image
    - The UArrays will have a length of blocksize * blocksize and the element size will the arrays will be passed by the client



Implementation Strategy:

1. Create a main function inside the uarray2b.c to run tests after each method for the struct is complete.
2. Implement our UArray2b_new and UArray2b_free functions
    a. Create an instance of a UArray2b in main() and free it after several assert statements which check if the UArray2b has been created properly
    b. Print out the number of columns and rows in the UArray2b to see if the proper space has been allocated in memory.
3. Implement our UArray2b_size, UArray2b_width, UArray2b_height, and UArray2b_blockSize

Implementation Plan for Part B (Implementation for the Manipulation of UArray2b):

1. Use a2blocked.c as a template for implementing the TO DO Functions in a2plain.c.

Implementation Plan for Part C (Actual Program to Rotate Images):

Read and process image to rotate

1. Open file and check if it's proper format/type
    a. If a file can't be opened, a message will print to stderr and the program exits w/ EXIT_FAILURE.
    b. Image will be in binary ppm format
    c. Use ppm.h to read and write
        i.) Relying on ppm interface to raise exceptions (do not need to catch)
            1. *Test: provide invalid files and make sure improper error messages are produced*

2.) Create a new instance of UArray2/UArray2b
    a.) Use A2 new to create new instance of UArray2b, get height and width and block size from pnmrdr map struct made when reading in
        1. *Test: assert that height, width, and blocksize match the values from the pnmrdr map struct*

Select what transformation to do

2. Process the user's specific input for a transformation
    a. If no transformation is selected default to a 0-degree rotation
        1. *Test: Run the transformation function w/o input from user and print messages that check if program has entered the 0-degrees transformation function*
    b. If 0 degrees, return the original image
        1. *Test: make sure original image prints*
    c. If 90 degrees, processing a row in the source image means processing a column in the destination image, and vice versa.
        i. Original [i][j] → transformed [h - j - 1][i]
            1. *Test: print out the row and column of the current pixel and double check the output to the mathematic calculations described above.*
    d. If 180 degrees, rows map to rows and columns map to columns
        i. Original [i][j] → transformed [w - i - 1][h - j - 1]
            1. *Test: print out transformed rows and columns, make sure nothing is out of bounds*
    e. When the user selects a transformation that's not implemented print an error message to stderr and exit w/ EXIT_FAILURE.
        1. *Test: make sure error is printed*

Return the new rotated image (stdout)

1.) Write the transformed image to output in binary ppm format

a.) Write an apply function that will print the image
b.) Magic number (P3), width, height, MaxVal, raster with newline at end of each line

2.) Free the old image using pnmrdr free
3.) Clean up UArray2b
4.) Exit with EXIT_SUCCESS

Time -

1. Create a new file to write to with fopen()
    a. The name given to the file will be what's after the -time flag.
2. Use a CPUTime_T object to time a transformation
    a. Start the timer immediately before the transformation starts and stop it right after
    b. Divide the time by the total number of pixels (A2Methods_UArray2's width * height)
3. Write it to the output file.
    a. *Test: After implementing the function for the -time flag, call the clock() function. Then, write a function to write the timing data to the specified file on the command line.*
    b. *Test: Specify a timing file that will be used and confirm the amount of time that this process should take.*
    c. *Test: Verify that this information is correct with the file contents of the outputted file.*
    d. *Test: Run the timer on different sized files. Ensure that time per pixel is similar if not same.*
4. Call the freeing function from the pnm.h interface to free heap-allocated objects
    a. Free any other objects in the heap

Part D Estimates

	row-major access (UArray2)	column-major access(UArray2)	blocked access (UArray2b)
90-degree rotation	3	3	4
180-degree rotation	1	1	2

Justifications

All of the 180-degree rotations were either first or second because they have better spatial locality than the 90-degree rotations because in 180-degrees, rows map to rows and columns map to columns, whereas in 90-degrees rows map to columns and vice versa. With a 180-degree rotation there is less jumping around in the cache since all the memory needed should be near each other. We did rank row and column major access as equal for both 90 and

180-degrees because the order in which memory is accessed should not matter. But, for blocked access, there is a chance that a block contains memory that is not relevant to the original image, whereas with row/col access, you know you have a higher chance of accessing memory from the original image.