Part C: ppmtrans, a program with straightforward locality properties

Opening file and processing information

Open and validate that the PPM file is valid through pnm.h functions (which also raises exceptions)

- If valid
    - Read file through pnm.h functions for reading, writing, and freeing the image file
        - Reading options:
            - From standard input
            - From file named on command line
        - Testing of file opening
            - Pass through properly formatted files and run echo $? to see the exit number
    - Create an applicable UArray2 or UArray2b based on pnm file, which has width and height dimensions, A2Methods_UArray2 (which can be any version of Uarray2) and an A2Methods_T which is a struct containing applicable UArray2 or UArray2b functions
    - Read file through Pnm_ppmread(FILE *fp, A2Methods_T methods), where fp is a FILE pointer, and methods is the applicable method struct based on the type of UArray
- If invalid input format, or file cannot be opened, exit through exit_failure and print explanatory message through stderr
    - Test
        - Pass through invalidly formatted files and run echo $? to see the exit number

QUESTION: for populating UArray2b based on how the Pnmrdr reads, do we need to populate through row major or block major if we are reading pixels line by line?

After opening file and processing information (performing transformation)

Step 1 (taking command input):

Call a2methods.h functions to perform transformation
Make function for rotation transformations

1. Will call different functions depending on parameter for degree transformation
    a. Apply_rotate_90
    b. Apply_rotate_180
    c. Apply_rotate_270
    d. If rotation is 0 → return

Make function for non rotation transformations

- Will call different functions depending on non-degree parameter
    - Timing operation, and optional flips and transpose transformations

Make function for row major, col major, and block major transformations

- Will call different functions depending on non-degree parameter
    - Modify_row_major
        - Copy pixels from the source image using map_row_major
    - Modify_col_major
        - Copy pixels from the source image using map_col_major
    - Modify_block_major
        - Copy pixels from the source image using map_block_major
    - Call methods->map_default without provided options

Step 2 (apply transformation):

Rotate 90:

- Column to row, row to column
- 0th column to 0th row
- Make function **apply_rotate_90(i, j, cl)**
    - Changes in pixels: (i, j) → (h - j - 1, i) where h is the height
    - Access height through int (*height) (A2 array2) function
    - Uses default mapping for parameterized A2 type
        - UArray2 default: row-major
        - UArray2b default: block-major
- Testing
    - For a small-scale 4x4 blocked array that has been mapped using block major with integer values 1-16, print out initial pixels in each corner before transformation, then after
        - Call UArray2b_at function passing through applicable array, column, and row parameters that correspond to corners

        - Initial top left should map to new top right, initial top right should map to new bottom right, initial bottom right should map to new bottom left, initial bottom left should map to new top left

Rotate 180:

- Row to row, column to column
- Flip upside-down
- Make function **apply_rotate_180(i, j, cl):**
    - Changes in pixels: (i, j) -> (w - i - 1, h - j - 1) where w is width and h is height
    - Access width through int (*width)   (A2 array2) function
    - Access height through int (*height)   (A2 array2) function
    - Uses default mapping for parameterized A2 type
        - UArray2 default: row-major
        - UArray2b default: block-major
    - Testing
        - For a small-scale 4x4 blocked array that has been mapped using block major with integer values 1-16, print out initial pixels in each corner before transformation, then after
            - Call UArray2b_at function passing through applicable array, column, and row parameters that correspond to corners
        - Initial top left should map to new bottom right, initial top right should map to new bottom left, initial bottom right should map to new top left, initial bottom left should map to new top right

Rotate 270:

- Row to column
- 0th row to 0th column
- Make **apply_rotate270(i, j, cl):**
    - Changes in pixels: (i, j) → (i, h - j - 1) where h is the height
    - Access height through int (*height)   (A2 array2) function
    - Uses default mapping for parameterized A2 type
        - UArray2 default: row-major
        - UArray2b default: block-major
    - Testing
        - For a small-scale 4x4 blocked array that has been mapped using block major with integer values 1-16, print out initial pixels in each corner before transformation, then after
            - Call UArray2b_at function passing through applicable array, column, and row parameters that correspond to corners

        - Initial top left should map to new bottom left, initial top right should map to new top left, initial bottom right should map to new top right, initial bottom left should map to new bottom right

Rotate 0:

- Remains the same
- Default operation if no ppm trans options
- Test similarly, printing out initial corner pixels before and after transformation to ensure that the image hasn't changed

For unimplemented commands, print a suitable error message to stderr and exit with a nonzero exit code.

Step 3 (writing based on X-major):

Row-major:

- Call row major function and map elements to output file
- Row major only applies to a UArray2
- Test through printing the transformed image to output

Col-major:

- Call col major function and map elements to output file
- Col major only applies to a UArray2
- Test through printing the transformed image to output

Block-major:

- Call Block major function and map elements to output file
- Block major only applies to a UArray2b
- Test through printing the transformed image to output

time <timing file>

- Follow a process similar to that described in https://www.cs.tufts.edu/comp/40/docs/locality-timing.html
- Include the correct files
- Create an open FILE with timeing_file name
    - Make sure it opens correctly
- Instantiate the timer
- Start the timer using CPUTime_Start(timer)
- End timer after rotation is complete with CPUTime_Stop(timer)
- Write to output file the time it took to complete the program.
- Testing for time function

        - Test time based on differently sized files to see if larger files take longer, also considering row, col, and block major functions

After performing transformation (writing to output)

- Call map_default()
    - Write each element to stdout in binary form
    - Pass through an apply function that will print/write output
- Write the new image using Pnm_ppmwrite(FILE *fp, Pnm_ppm pixmap);
    - Clarify this point… it doesn't seem like we are writing to a file, so is this needed?
    - Regardless, we will use this for testing to open up transformed files and analyze output
- Free the UArray2 or UArray2b, and the image that was created, by using Pnmrdr_free and passing through the image
- After successfully writing image, exit with exit_success

Part D (experimental): Analyze locality and predict performance

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

In a UArray2b, The blocked elements will have the same hit rate in each case only two blocks will need to be cached at the same time, which means there will be less misses. It is worse than the mapping row major 180 degree rotations because at most blocksize elements will be stored sequentially in memory, while as row major access at most array widths will be stored sequentially. It is better than the column major mapping because it has greater spatial locality as block size amount of columns are stored nearby in memory, whereas column major mapping columns are not stored nearby in memory. It is better than 90 rotations row major for a similar reason. It preserves the spatial locality because at most 2 blocks are used for the 90 degree rotation every blocksize time. With row major and column major 90 degree rotations the program jumps around rows each iteration, making it not very spatially local as rows are stored sequentially.

In a regular UArray2, 180 degree rotation by row major will have a better hit rate than 18 degree transformations through column major. This is because rows are stored sequentially in memory. In row major we iterate over the entire row before moving on to the next row. This means that after the first access, the cache will only hit, making this faster than other methods. In column major mapping we are going to each element in the column, skipping rows, which is bad for spatial locality, making it slower than row major 180 degree transformations. With 90 degree transformations, row and column major access will have the same cache misses as they will both be required to iterate through two elements at a time, which results in a greater amount of misses, and these will be slower than both 180 degree transformations as a result.