Part A: Design Checklists for ADTs:

1. What is the abstract thing you are trying to represent?

The abstract thing we are trying to represent is a 2D array of any object type

2. What functions will you offer, and what are the contracts those functions must meet?

extern T UArray2b_new (int width, int height, int size, int blocksize)

Inputs:

- int width: the width of the 2d array (number of columns)
- int height: the height of the 2d array (number of rows)
- int size: the size of each index in the 2darray
- int blocksize: the number of cells on one side of a block

Outputs:

- A UArray2b object

Does:

- Creates a 2d array of width, height with indices of size. The array is stored in memory as blocks whose size is determined by blocksize.

extern T UArray2b_new_64K_block(int width, int height, int size);

Inputs:

- int width: the width of the 2d array (number of columns)
- int height: the height of the 2d array (number of rows)
- int size: the size of each index in the 2darray

Outputs:

- A UArray2b object

Does:

- Creates a 2d array of width, height with indices of size. The array is stored in memory as blocks whose size is default. Blocksize is defaulted to largest as possible while still allowing a block to fit in 64KB of RAM. If 1 cell will not fit in 64KB, blocksize = 1.

extern void UArray2b_free (T *array2b)

Inputs:

- T* array2b: pointer to a UArray2b object

Outputs:

- none

Does:

- Frees the memory associated with the object

extern int UArray2b_width (T array2b);

Inputs:

- T array2b: a UArray2b object

Outputs:

- int: the width of the object

Does:

- Returns the width of the 2darray

extern int UArray2b_height (T array2b);

Inputs:

- T array2b: a UArray2b object

Outputs:

- int: the height of the object

Does:

- Returns the height of the 2darray

extern int UArray2b_size (T array2b);

Inputs:

- T array2b: a UArray2b object

Outputs:

- int: the size of the indices of the object

Does:

- Returns the size of the indices of the object

extern int UArray2b_blocksize(T array2b)

Inputs:

- T array2b: a UArray2b object

Outputs:

- int: the number of cells on 1 side of a block that the memory is stored in

Does:

- Returns the blocksize of the object

extern void *UArray2b_at(T array2b, int column, int row)

Inputs:

- T array2b: a UArray2b object
- int column: the column index of the value
- int row: the row index of the value

Outputs:

- void *: a void pointer to the object that is stored in the array at row, col

Does:

- Returns a pointer to the object at row,col

```
extern void UArray2b_map(T array2b,
                         void apply(int col, int row, T array2b, void *elem, void *cl),
                         void *cl);
```

Inputs:

- T array2b: a UArray2b object
- void apply(int col, int row, T array2b, void *elem, void *cl): a pointer to the function that wants to be applied
- void *cl: closure value for the apply function

Outputs:

- none

Does:

- Maps the apply function to all values in the UArray2b object.

3. What examples do you have of what the functions are supposed to do?

```
UArray2b_T arr = UArray2b_new (2, 2, 4, 2);  //creates 2x2 array, indices are 4 bytes, memory
                                                // blocksize=2
int wid = UArray2b_width (arr);      //wid is set to 2
int hi = UArray2b_height (arr);      //hi is set to 2
int size = UArray2b_width (arr);    //size is set to 4
int b_size = blocksize (arr);         //b_size is set to 2

int row = 1;
int col = 1;
int *p = Array_at(arr, col, row);     //capture into array at row,col
assert(size== sizeof(*p));         //make sure pointer sizes match
*p = f();                             // set the value of p equal to the result of some function

UArray2b_map(arr, print_func, NULL);  //apply some print function to all values
UArray2b_free (T &arr);  //frees the memory for the object
```

4. What representation will you use, and what invariants will it satisfy?

We will represent a box as a single UArray_T as the specification suggest. Each of these blocks will be stored in a UArray2_T, where each element is one UArray_T block. An UArray2b_T can be represented as an UArray2_T, each element of which contains one block (a UArray_T).
Since UArray_T returns a contiguous array of bytes, we know that each element in the UArray2_T block are in nearby memory locations.

As stated in the specification, we can find cell indexed at (i, j), by first finding the block at index
(i / blocksize, j / blocksize).

To determine the location of any index (col, row) in UArray2_T, such that a UArray2_T object (e.g uArray2 (and each UArray2_T contains a UArray_T (e.g uArray))) would have element (col, row) at uArray2->uArray[X] where X is determined by the function depicted below.

```
if (currRow == 0 && currCol >= 0)
        return currCol;
else
        return ((colWidth*currRow) + currCol);
```

This accesses the UArray_T block stored in UArray2_T.

Then to access the correct index within that block we can do:
(blocksize * (i % blocksize) + j % blocksize).
This will give you the location of (col, row) in a UArray_T block.

5. How does an object in your representation correspond to an object in the world of ideas?

The main object in our representation is UArray2b_T, this “represents” all the data stored.

The UArray2b_T in our representation is powered by UArray_2, which stores, in a single contiguous array, a “2d array”. Each element in this 2d array corresponds to a “block” from the input data, a block contains elements from neighboring rows and columns, for example the first 4x4 block would contain the elements from the data at all indexes in the range: row: 0-3 & col: 0-3. These indexes from the input data will be stored next to each other in memory by using a UArray_T.

Once these elements have been stored in the UArray_T, the array now represents a full block and is stored in UArray2_T, once all lines from the inputted data are parsed and all rows and columns are “converted and saved” to blocks, the UArray2_T now represents the full data set, which is the UArray2b_T.

6. What test cases have you devised?

- Index out of bounds
- Make size larger than block size
- Use UArray2b_new_64K_block with a very large size parameter (should default to blocksize=1)
- Use UArray2b_new_64K_block with a normal size parameters(should default to largest possible in 64 KB)
- UArray2b with width/height/size/blocksize = 0
- Create a print apply function to see if the mapping function and data storage is correct
- Input a very large file
-

7. What programming idioms will you need?

- Getting rid of "Unused variable" warnings
- Idioms for void ** pointers
- Type abbreviations for structure types
- Using an abstraction defined in an interface Foo
- Handling void * values of known type
- Using unboxed arrays
- Initializing array elements
- Allocating memory

Part C: Design Checklists for writing programs:

1. What problem are you trying to solve?

Implement a program that performs specified image transformations.

2. What example inputs will help illuminate the problem?

Images that need to be transformed in different ways

3. What example outputs go with those example inputs?

The resulting transformed image and timing data of the transformation.

4. Into what steps or subproblems can you break down the problem?

1. Reading data from input file
    a. Opening file
    b. Reading element by elements
2. Saving data from input file
    a. As each element is read, save in corresponding location
3. Transforming image according to desired outcome
    a. Determine transformation types
        i. 0 & 90 & 180 degrees
    b. Exexecuate transformation
4. Outputting transformation

5. What data are in each subproblem?

1. Filename and pointer to a file. Each pixel will be iterated through.
2. A2Methods_T object
    a. Will contain either UArray2_T or UArray2b_T object

6. What code or algorithms go with that data?

The methods suite will be used to call functions for the 2d array that is used. Mapping functions for the arrays are important algorithms that will be used in order to apply functions to all elements in a 2d array in a certain order. Row and column major mapping functions will be used to call these function. Algorithms are provided to read data in.

7. What abstractions will you use to help solve the problem?

A2Methods which uses UArray2 and UArray2b.
UArray2 and UArray2b use UArray

8. If you have to create new abstractions, what are their design checklists?

N/a -- UArray2b checklist above

9. What invariant properties should hold during the solution of the problem?

The number of indices in the 2darray should not change. The block size for a UArray2b object should not change either. This is because although the width and height may switch due to a transformation, there will not be data added or subtracted from the array. The block size for the blocked 2darray will also not change, because the transformation will not change the way that the memory is stored.

The method for indexing into a cell will also be constant. To index a cell, the blocksize is calculated using user inputs i and j. i/blocksize, j/blocksize gives the block that the cell is in. Then to get the index in this array that the cell is located in, the equation blocksize* (i% blocksize) + j % blocksize is used.

10. What algorithms might help solve the problem?

The important algorithms are the transformation algorithms that allow the data in the 2d arrays to be transformed by the input on the command line. These algorithms will be functions that can be called as function pointers by the mapping functions. There will also need to be a printing/writing function that is called by mapping functions to output the result, this function must with with row-major, col-major and block-major calls.

11. What are the major components of your program, and what are their interfaces? Components include functions as well as abstract data types. An interface includes contracts as well as function prototypes.

A2Methods_T, UArray2b_T, UArray2_T, UArray_T are the main ADTs.
A2Methods suite has all of the functions that are contracted below to be used on the 2darray classes to perform transformations. UArray_T is the underlying data structure to make the 2d arrays work.

void rotate90(int row, int col, void *elem, void *cl);

- Inputs:
  - int row: the current row of the array
  - int col: the current column of the array
  - void *elem: pointer to the element that we are currently looking at
  - void *cl: closure value, will be a new 2darray object
- Outputs:
  - None
- Does: calls transpose and then flip horizontal function in order to rotate 2d array 90 deg

void rotate180(int row, int col, void *elem, void *cl);

- Inputs:
  - int row: the current row of the array
  - int col: the current column of the array

        - void *elem: pointer to the element that we are currently looking at
        - void *cl: closure value, will be a new 2darray object
    - Outputs:
        - None
    - Does: calls flip vertical then flip horizontal to rotate image 180 deg

void rotate270(int row, int col, void *elem, void *cl);

    - Inputs:
        - int row: the current row of the array
        - int col: the current column of the array
        - void *elem: pointer to the element that we are currently looking at
        - void *cl: closure value, will be a new 2darray object
    - Outputs:
        - None
    - Does: calls transpose and then flip vertical function in order to rotate 2d array 270 deg

void transpose(int row, int col, void *elem, void *cl);

    - Inputs:
        - int row: the current row of the array
        - int col: the current column of the array
        - void *elem: pointer to the element that we are currently looking at
        - void *cl: closure value, will be a new 2darray object
    - Outputs:
        - None
    - Does: fills new array by putting elem in [row][col] instead of [col][row]

void flip_horizontal(int row, int col, void *elem, void *cl);

    - Inputs:
        - int row: the current row of the array
        - int col: the current column of the array
        - void *elem: pointer to the element that we are currently looking at
        - void *cl: closure value, will be a new 2darray object
    - Outputs:
        - None
    - Does: flips the 2d array by moving elem from [col][row] to [width-1-col][row]

void flip_vertical(int row, int col, void *elem, void *cl);

    - Inputs:
        - int row: the current row of the array
        - int col: the current column of the array
        - void *elem: pointer to the element that we are currently looking at

    - void *cl: closure value, will be a new 2darray object
- Outputs:
    - None
- Does: flips the 2d array by movin gelem from [col][row] to [col][height -1 - row]

12.How do the components in your program interact?
That is, what is the architecture of your program?

A image file is read, written and freed using pnm.h. The data from this image file is put into a UArray2 or UArray2b object. Then the A2Methods suite is used to call any functions and transformations that need to be called on the 2darray object. The result of the transformations will be used to print the file.
Also, the 2d array objects use UArray_T objects. For a UArray2_T, the 2darray is actually made up of a 1d array of height*width indices. The UArray2b_T object uses a UArray2_T objects to represent the 2d blocks while UArray_T objects are the objects inside each index.

The transformation functions that are talked about in #11 are functions in the A2Methods suite that are called on 2d array by using function pointers that are called by the methods suite. The functions that are to be called are determined by command line inputs and the data is also determined through the command line.

13. What test cases will you use to convince yourself that your program works?

Input different images and manually view the output that comes out of the transformation.

14. What arguments will you use to convince a skeptical audience that your program works?

Run different transformation on images and show their outputs which will allow a visual to see that the program can properly transform images.