CSC 110: Fundamentals of Programming I
Assignment #6: 2-Dimensional arrays

Due date

Sunday, November 22nd, 2015 at 11:55 pm via submission to connex.

How to hand in your work

Submit the file **`ImageManipulate.java`** in the Assignment #6 link on connex.

Learning outcomes

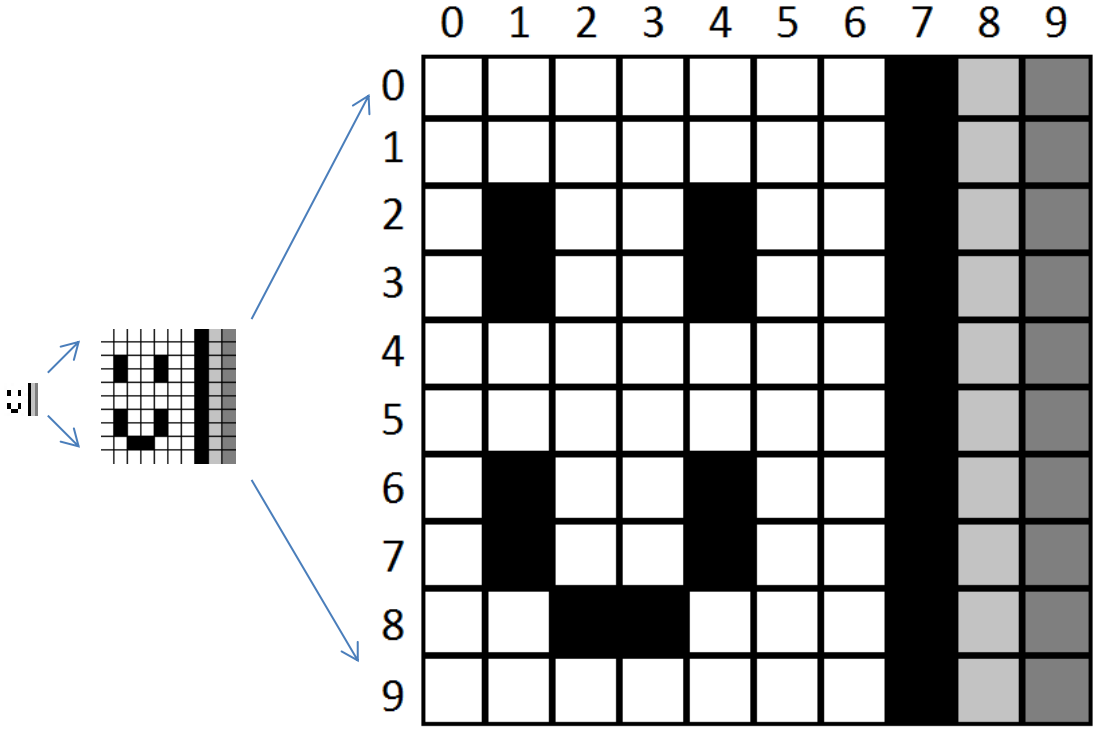
When you have completed this assignment, you will understand:

- How to pass *parameters* and *return* values using `static` methods.
- How to pass arguments from the command-line to the `String[] args` array
- How to use *two-dimensional* `(2D) arrays.`
- How to orient yourself with and extend previously written code.
- How to *indent* and *document* a Java program.

One well-known set of tasks for computers today is the manipulation of images. Your work for this assignment is to add additional methods to the program **`ImageManipulate.java`**. Each execution of the program will perform either one output transformation (ASCII-art version of image) or an image manipulation (i.e., reflect across x-axis; reflect across y-axis; tile an image; invert image, rotate image).

For this assignment, you will be using the command-line interface to pass in the names of the input and output files, as well as the image manipulation operation information. **Solutions must not use a Scanner object**, or require any other interaction from the user during execution.

To illustrate how the data in an image file is represented in a 2D array of integers, consider the very small image below that is 10 pixels wide by 10 pixels high below:



If we really zoom in on this image, we can see that the image is a matrix of pixels, each representing some shade of gray. A 2D array of integers with values ranging from 0 to 255 to represent the above image could be constructed in our program. Given the image above named "example.png", using the **`readGrayscaleImage`** method provided, we could create a 2D array with the same values shown above.

```
int[][] image = readGrayscaleImage("example.png");
```

The above statement would create an array with the same values as the following:

```
int[][] image = {
      {255, 255, 255, 255, 255, 255, 255, 000, 195, 126},
      {255, 255, 255, 255, 255, 255, 255, 000, 195, 126},
      {255, 000, 253, 255, 000, 255, 255, 000, 195, 126},
      {255, 000, 255, 255, 000, 255, 255, 000, 195, 126},
      {255, 255, 255, 255, 255, 255, 255, 000, 195, 126},
      {255, 255, 255, 255, 255, 255, 255, 000, 195, 126},
      {255, 000, 255, 255, 000, 255, 255, 000, 195, 126},
      {255, 000, 255, 255, 000, 255, 255, 000, 195, 126},
      {255, 255, 000, 000, 253, 255, 255, 000, 195, 126},
      {255, 255, 255, 255, 255, 255, 255, 000, 195, 126}
};
```

As you can see in the 2D array of integers above, all of the black pixels in the image have a value of 0, all of the white pixels in the image have a value of 255, and all the gray pixels have a value in between, depending on how light or dark they are.

Recommended steps to follow in order

1. Similar to Assignments 4 & 5, the specification document outlines all of the required methods for this assignment. Appendix A illustrates some examples of how to call and test the methods in your ImageManipulate.java program.

2. In this assignment the `args` array in the `main` method specifies the name of the input image file to read and output image file to create. It also specifies which method to run. Complete the main method first, by calling the `readGrayscaleImage` method, which creates a 2D array of integers from an image file. The array is what you will use in the manipulation methods.

3. Now start working on the manipulation methods. Start with the `invert` method. This method returns a 2D array back to main with all values inverted. When this is complete, you can pass the resulting 2D array into the `writeGrayscaleImage` method, and see the inverted image!

Marking

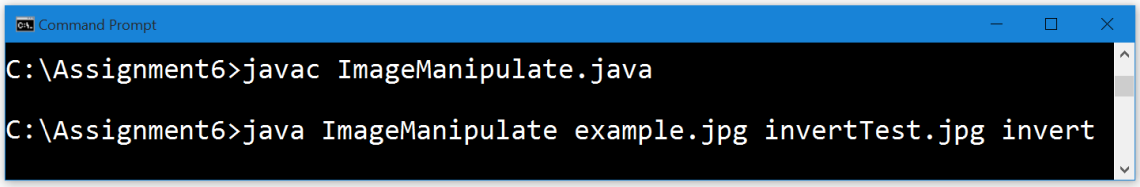
Your mark will be based on the following criteria:

- Your code *must compile and run*. Some examples of how to test your methods, along with expected output, are outlined in **Appendix A.**
- Your code must conform to all the requirements mentioned in the specification document.
- Test each of the required methods to ensure each one functions correctly.
- Your code must follow the guidelines outlined in Style_Guidelines.pdf, found through the Lectures & Stuff link in the Lab Resources folder on connex. You may notice that the specification document provides some very nice comments you are welcome to borrow.

Appendix A – Testing your code

Getting started setting up the *main* method:

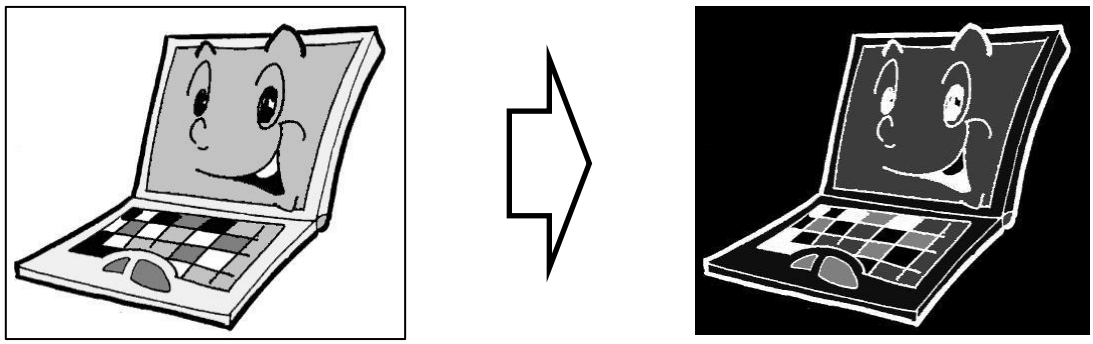
All of the information needed by the program will be passed in through the `args` array when the program is executed, there will be **no use of Scanners**. Let's assume the user compiled and executed the program the following way:



This will put the following contents into the `args` array in the `main` method:

`args: {"example.jpg","invertTest.jpg","invert"} args.length: 3`

This command tells your program to use the data in the **example.jpg** file, **invert** it, and create a new image file **invertTest.jpg** with the inverted values.



example.jpg invertTest.jpg

The program will always be executed in the same fashion:

`java ImageManipulate <inputFile> <outputFile> <operation>`

As shown above, the first argument is the input file's name, the second argument is the output file's name, and the remaining argument(s) are used to call the corresponding method in the program (the tile method actually requires 5 arguments).

In your main method, first ensure that args array has at least 3 elements in it. After this is done, you can begin to set-up your variables. Using the input file name (first argument), you can call the method `readGrayscaleImage` to create a 2D array of integers. Based on the operation name (third argument), one of the methods in your program will be called. Each method manipulates the values in a 2D array of integers and returns a new array. This new array can be passed into the `writeGrayscaleImage` along with the output file name (second argument) to create a new image file. Open up the new file to see the results!

Testing the *invert* method:

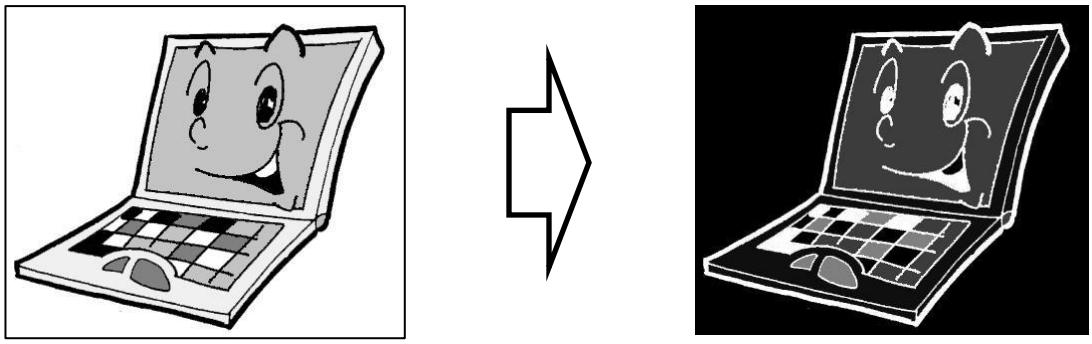
Given there is an image file called "computer.jpg" in the same folder as the Java file, to create a new image file called "output.jpg" we would run our java program the following way:

```
java ImageManipulate computer.jpg output.jpg invert
```

Tips:

1) The output image has the same dimensions (height and width) as the input image (so the 2D arrays of integers will be the same size)
2) Inverting a number of a 10-point scale might be easier to think about (0 becomes 10, 1 becomes 9, 2 becomes 8, etc). For this case, we want to invert numbers on a 255-point scale. What is the pattern?

Given the file on left is "computer.jpg", the following file will be created:



Testing the *makeAscii* **method:**

Given there is an image file called "computer.jpg" in the same folder as the Java file, to create a new image file called "ascii.txt" using a **PrintStream**, we would run our java program the following way:

```
java ImageManipulate computer.jpg ascii.txt makeAscii
```

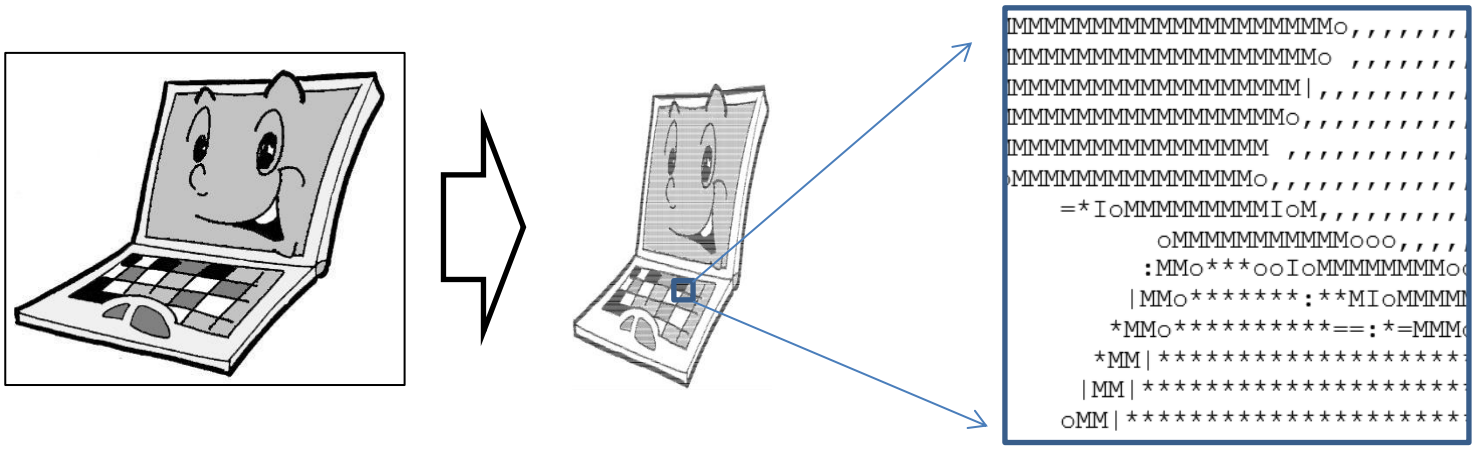
NOTE: For this operation the writeGrayscaleImage method is **not** called

Tips:

1) Create a PrintStream to write each ASCII character into. Each character is based off of the integer value in the 2D array, using the following integer value to character mappings:

Value	Character
0 to 20	M
21 to 40	L
41 to 60	I
61 to 80	o
81 to 100	|
101 to 120	=
121 to 140	*
141 to 160	:
161 to 180	-
181 to 200	,
201 to 220	.
221 to 255	(space)

Given the file on left is "computer.jpg", the following text file will be created:



Testing the `tile` method:

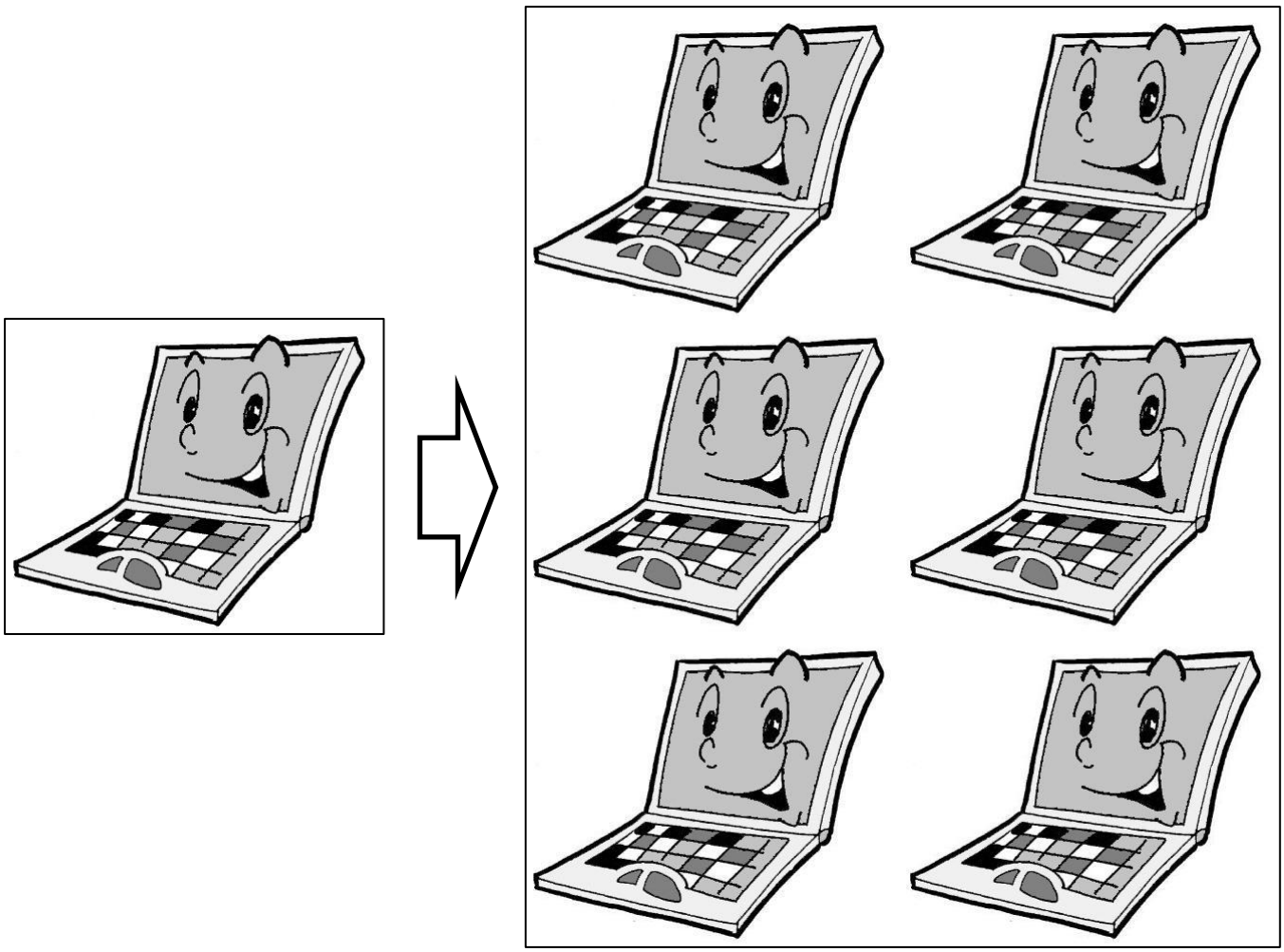
Given there is an image file called "computer.jpg" in the same folder as the Java file, to create a new image file called "output.jpg" we would run our java program the following way:

```
java ImageManipulate computer.jpg output.jpg tile 2 3
```

Tips:

1) The output image's height and width will be a multiple of the originals. If the parameters passed in are 5 and 7, the output image's width will be 5x the original image's width, and 7x the height of the original image.
2) Each of the tiled images will be an exact replica of the original image.

Given the file on left is "computer.jpg", the following file will be created:



Testing the `verticalMirror` method:

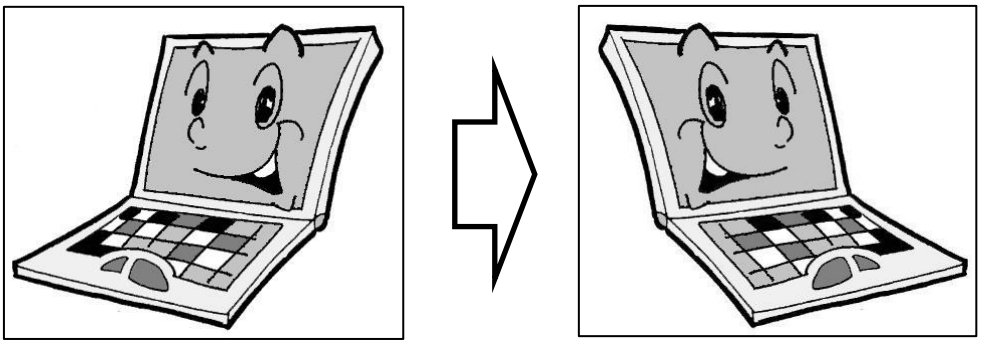
Given there is an image file called "computer.jpg" in the same folder as the Java file, to create a new image file called "output.jpg" we would run our java program the following way:

```
java ImageManipulate computer.jpg output.jpg verticalMirror
```

Tips:

1) This method reverses all of the values across each row in a 2D array. Think about how to reverse the values in an array.
2) Each value does not move up or down a row, only its column number is changed.

Given the file on left is "computer.jpg", the following file will be created:



Testing the *horizontalMirror* **method:**

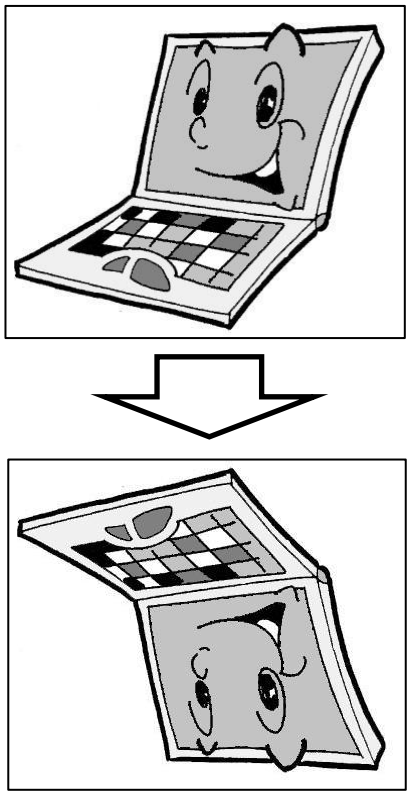
Given there is an image file called "computer.jpg" in the same folder as the Java file, to create a new image file called "output.jpg" we would run our java program the following way:

```
java ImageManipulate computer.jpg output.jpg horizontalMirror
```

Tips:

1) This method reverses all of the values down each column in a 2D array.
2) Each value does not move left or right to a different column, only its row number is changed.

Given the file on left is "computer.jpg", the following file will be created:



Testing the `rotate` method:

Given there is an image file called "computer.jpg" in the same folder as the Java file, to create a new image file called "output.jpg" we would run our java program the following way:

```
java ImageManipulate computer.jpg output.jpg rotate
```

Tips:

1) If the image is not square, the output image's height with be equal to original image's width, and width equal to the original image's height.
2) Try creating a 2D array on paper and mapping out where each value in the array goes during a rotate. Try and find a pattern, and this should help with setting up your algorithm that you can use in your solution.

Given the file on left is "computer.jpg", the following file will be created: