Eric Roberts and Jerry Cain
CS 106J

Handout #39
May 17, 2017

Section Handout #6: [arrays][pt2]

Portions of this handout by Eric Roberts

1. 2D Array Warmup

In spreadsheet programs like Excel or Google Sheets, they have functionality that allows you to sum a row, sum a column, and sum all. Given a 2D array **`sheet`**, please implement the following functions:

```
function sumRow(sheet, rowNum)
function sumCol(sheet, colNum)
function sumAll(sheet)
```

Remember to return the string **`"Error: [row/col] index [indexNum] out of bounds"`** if the indices **`rowNum`** or **`colNum`** is out of bounds of **`sheet`**. For example:

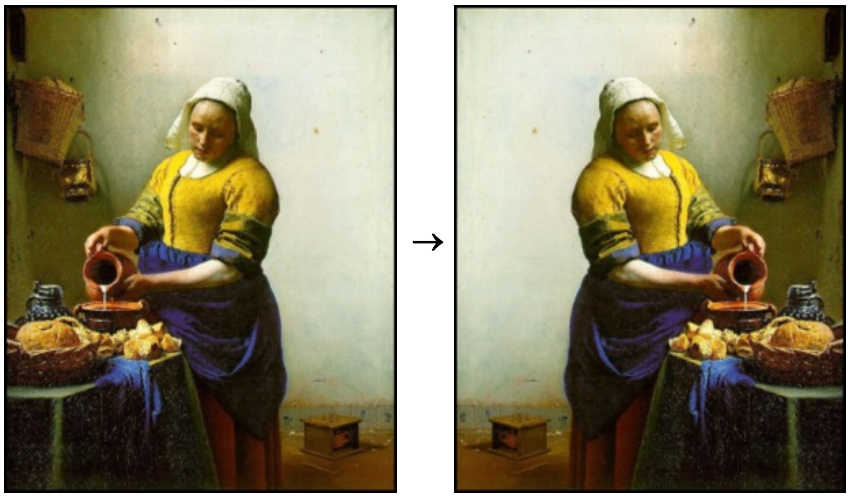
```
var sheet = [[1, 2, 3],
             [4, 5, 6],
             [7, 8, 9]]

sumRow(sheet, 1) //returns 15
sumCol(sheet, 0) //returns 12
sumAll(sheet) //returns 45
sumRow(sheet, 3) //returns "Error: row index 3 out of bounds"
```

Out of this warmup, you should have an understanding of the difference between **`array.length`** and **`array[0].length`** for 2D arrays and when to use each.

2. Image processing

Write a function **`flipHorizontal`** that reverses a picture in the horizontal dimension. Thus, if you had a **`GImage`** containing the image on the left (of Jan Vermeer's *The Milkmaid*, c. 1659), calling **`flipHorizontal`** on that image would return a new **`GImage`** as shown on the right:



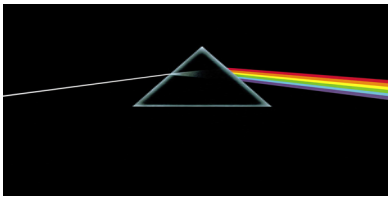
3. Image scale

We learned in lecture about the **`scale`** function that comes with **`GImages`** and the graphics library. However, Jerry never told us how it worked behind the scenes. Let's find out!

Write a function **`scale`** that takes a **`GImage`** and two **`ints`** representing the scale factors **`scale_x and scale_y`** in the x and y directions, respectively. For example, given this original image from the cover of Pink Floyd's "The Dark Side of the Moon", the result of the following function calls should be:

`original aka scale(image, 1, 1)`

`scale(image, 2, 1)`



`scale(image, 0.5, 2)`

`scale(image, 2, 2)`

Perhaps we can simplify our approach by just considering the scaling in one direction. Once we figure out how to do that, we can extend our functionality to include both axes.

Lastly, to simplify the problem even further, you can assume you have a helper function **`create2DArray(rows, cols, value)`** that creates a 2D array of size **`rows, cols`** where each element is initialized to **`value`**.

4. How Prime! [Back to 1D]

In the third century B.C., the Greek astronomer Eratosthenes developed an algorithm for finding all the prime numbers up to some upper limit *N*. To apply the algorithm, you start by writing down a list of the integers between 2 and *N*. For example, if *N* were 20, you would begin by writing down the following list:

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

You then begin by circling the first number in the list, indicating that you have found a prime. You then go through the rest of the list and cross off every multiple of the value you have just circled, since none of those multiples can be prime. Thus, after executing the first step of the algorithm, you will have circled the number 2 and crossed off every multiple of two, as follows:

From here, you simply repeat the process by circling the first number in the list that is neither crossed off nor circled, and then crossing off its multiples. Eventually, every number in the list will either be circled or crossed out, as shown in this diagram:

The circled numbers are the primes; the crossed-out numbers are composites. This algorithm for generating a list of primes is called the sieve of Eratosthenes. Write a program that uses the sieve of Eratosthenes to generate a list of all prime numbers between 2 and 1000.

Bonus: Y'all are amazing!

Give yourself and the people around some positive affirmations! You're doing awesome and you've all learned so much since day 1 with Karel! ☺ Also to all the SLs who are teaching this section, thanks for everything you do!