CSci 1103 Fall 2018 Lab 6 – Arrays

Lab 6 will include:

- Arrays
- 2-dimensioned Arrays

Please refer to the **Lab Procedures for the Semester** in the Lab 1 writeup as they pertain to all labs for this course.

At any point, if you have questions, please *ask a TA to help you*. That is what they are here to do.

Step 1. Building A Histogram Array and Initializing It.

Note: The Canvas Lecture Examples section for Week 5 contains `Histogram.java`*. You may find it helpful to look at how some similar things are done in that code, but please do not directly copy it because what is required for the steps in this lab is different from the posted* `Histogram.java` *code. Additionally, if you do take ideas from* `Histogram.java`*, be sure to credit this with a comment inside your code.*

Begin by creating a new directory for this lab within your home directory. Call it lab6. Within the lab6 directory, create a new file called `MyHistogram.java`.

We will be writing a program that will take integer quiz scores in the range of 0 to 100 as command line arguments and put them into an array to be used as a histogram. Begin by declaring and creating within your program an array of `int`s that is indexed from 0 to 100 (inclusive) using the `new` operator. Call it `hist`. How long should this array be?

Now write a loop to initialize `hist` to all zeroes. Follow it with a *second* loop to print out *only* the values contained in the array that are *non*-zero prefaced by their corresponding indices. For now, since all entries are zero, the second loop will not print anything.

Important note: the posted `Histgram.java` *code prints out a row of stars for each non-zero entry, where the number of stars corresponds to the non-zero value stored in the histogram at an index. Your program will only need to print out an integer rather than the row of starts. However, feel free to implement both ways of displaying the histogram information even though this lab will use the numerical format.*

Compile your code to make sure that it is syntactically correct, and run it. There should be no output, but there should be no run-time errors either.

Have this step checked by a TA.

Step 2. Loading up the Histogram.

One of the nice things about histograms is that they keep track of the *number of times* each value occurs, rather than the individual values themselves. This means that we always know how big to make the array if we just know the *range* of the scores. Otherwise stated, we do *not* need to know how *many* scores there will be, just what the *range* of scores will be. Histograms also allow us to keep virtually all the information that would be available if each individual score was saved, while maintaining a constant amount of space. This is important because if we chose instead to keep each individual score, we would need to have a way of handling the situation where the number of scores is greater than the amount of array space we declared--and the amount of space required could be much greater than if a histogram were used.

Recall that command line arguments go into an array which we typically refer to as `args`. The runtime java system is very good at sizing that array to match the number of arguments that are entered. The size of any array can be accessed through the `length` attribute. So, `args` will contain each individual score entered on the command line (some of which could be erroneous), and `args.length` will tell us how many scores there are in `args`. In this step, we will build the histogram by taking each valid score in args and updating the value found at the corresponding index in the array `hist`. But, be careful of two things:

1) Values stored in `args` are `String`s, so they need to be converted to `int`s as we have done before
2) While you *can assume* that all values entered on the command line will be integers, you can*not* assume the values are within the range of 0-100. So, be sure to verify that a value is in range before attempting to update `hist`.

Write a loop (and locate it between your initialization and printing loops in your code from Step 1 above) that will process each legal score from `args` by updating the appropriate index in `hist`. Compile and run your program. With values entered into your histogram, you should now see some output. When you are convinced that your code is working, have a TA check this step.

Along with other tests you run, the TAs will want to see the results of the following test case:

```
java MyHistogram 5 7 99 89 4 5 91 23 56 99 85 87 17 18 23 99 100 91 72 72 72 99
```

Step 3. Gathering Statistics.

Once the scores are represented in `hist`, we can gather statistics. Let us start by finding the *lowest* score and the *highest* score. Write code to do this within `MyHistogram` and display the results. Test your code, and pay attention to boundary conditions. Have this step checked by a TA.

Step 4. More Statistics.

Now add code to your program to find *two* more statistics: The *average* and the *mode*. The average is the same as the mean, and the mode is the score that occurs most frequently. Mode can be tricky because there can be *more than one* mode, and you should print all of them. (The mode can be assumed to be zero for an empty histogram.) Test your code and have this step checked by a TA when ready.

Step 5. Extra Credit: Median.

For some extra practice and extra credit, add code to your `MyHistogram` program to find the *median* score. The median is the middle score so that the median is >= ½ the scores and the median is also <= ½ the scores. In the case where there is an even number of scores *and* no score at the median, average the first existing score above the median with the first existing score below the median. Since this step is optional, you may choose to go directly to Step 6, and possibly return to Step 5 later.

Step 6. To The Next Dimension.

In this step, we will write a program to print out a simple character-based image frame that contains random picture elements (pixels). While we outline the basic substeps, quite a bit is left to you to engineer.

Begin a new program, call it `Picture.java`. `Picture.java` will take in *three* command line arguments—*all positive integers*. The first will be the number of rows (`rows`), the second will be the number of columns (`cols`), and the third will be the number pixels to set (`npix`).
Follow the substeps below to complete this step:

1. Declare a `main()` method that contains a two-dimensional array of `char` like this: `char[][] frame;` Then, verify that both `rows` and `cols` are > 1. If the values are valid, allocate the array to be of size: `int[rows][cols]` using the `new` operator.

2. Initialize the entire array to all spaces using a nested loop. Verify that your program compiles without error at this point before you continue to the next substep.

3. Using the third command line argument, `npix`, write a loop that will make `npix` attempts to *randomly* place a '*' character in your frame array. To randomly select a location in your frame array, you can make two calls to the `randomInt(min, max)` number generator from Project 2. Copy and paste or just rewrite the `randomInt()` code and put it in your `Picture.java` file just before your `main()` method. The first call to `randomInt()` will give you an index for the first dimension, the second call will give you an index for the second dimension. *Note: Since you are generating random locations within your frame, it will often be the case that the same indices will be generated multiple times. This is not a problem, but it means that you should expect to have fewer than `npix` '*' characters in the frame array.* Verify that your program compiles after adding in this code and the `randomInt()` method.

4. Near the end of your `main()` method, write a nested loop to display the `frame` array to the screen. Keep in mind that `println()` will start a new line after printing, and `print()` does not start a new line after printing. Try your code on a variety of sizes (values for `rows` and `cols`) and values for `npix`.

Have this step checked by a TA.

That's it! Have a great week!