**Mahmut Alp Erkent**

**5.1.2020**

# Assignment #3

## 1) Problem description:

The problem asks for a Java program that reads a PPM file as an input and outputs either the same content, grayscale version, convoluted version (using content of another file as a filter) or color-quantized version, deciding what to do according to the arguments fed to the program.

## 2) Problem solution:

To solve this problem, my program had to read a PPM file, create a 3D array that stored the RGB values of every pixel, then do some manipulations on the values of the array and finally, create a new file and output the values of the array to that file. Therefore, I created different methods for reading, writing, grayscaling, convoluting and quantizing (and a few more for practical reasons).

In the **main** method, I read the arguments first and created a scanner to read the input file and a 3D array using that scanner as a parameter for the *readPPM* method. Then, I used if/else if conditionals to decide the behavior of the program according to the mode argument. I created PrintStream objects and used the *writePPM* method to print out the corresponding file in all conditionals and also created extra variables to read the third argument for mode 2 and 3.

**Part 1:** In the **readPPM** method, a scanner is used as a parameter to read the input file and extract the RGB values of every pixel. Then, a 3-D array is created and the value of every element is assigned via three (1+(1+(1)) nested for loops, returning the array to the *main* method.

**Part 2:** In the **writePPM** method, a PrintStream and a 3-D array are used as parameters to write the value of every element in the array to the output file via three (1+(1+(1)) nested for loops.

**Part 3:** In the **grayscale** method, a 3-D array is used as a parameter to average out the RGB values of every corresponding pixel and writing the averages into a new array via four (1+(1+(1+1))) nested for loops, which is then returned to the *main* method.

**Part 4:** In the **convolution** method, a 3-D array and a scanner are used as parameters to read the filter file and extract the values of the kernel matrix. Then, a 2-D array is created and the value of every element is assigned via two nested for loops. Later, a 3-D array is created using the dimensions of the parameter array and the filter array, and the value of every element is calculated via five (1+(1+(1+(1+(1)))) nested for loops. After the convolution operation is done, the resulting value is checked, reduced to the range of 0 to 255 via if/else if conditionals and assigned to the corresponding element. Finally, the convolved array is returned to the *main* method.

**Part 5:** For quantization part, four different methods are used:

In the **quantization** method, a 3-D array and an integer indicating the range argument are used as parameters to create a copy of the array to be used in the method and a new boolean array of the same size to keep track of quantized elements. Then every element is quantized in specified order via three

(1+(1+(1))) nested loops by calling the *quantizer* method. Finally, the quantized array is returned to the *main* method.

In the i**sValid** method, a 3-D array and three integers indicating the indices of the element are used as parameters to check if the indices are within the array and a boolean output is returned to the *quantizer* method.

In the **inRange** method, a 3-D array and five integers (three indicating the indices of the element, one indicating the quantizer value and last one indicating the range argument) are used as parameters to check if the value of the element at given indices are within the range of the quantizer value, and a boolean output is returned to the *quantizer* method.

In the **quantizer** method, two 3-D arrays (an integer array storing RGB values of pixels and a boolean array keeping track of quantized elements) and five integers (three indicating the indices of the element, one indicating the quantizer value and last one indicating the range argument) are used as parameters to first mark the element at given indices as true in the boolean array, and then using six if conditionals, to quantize each neighboring cell in specified order if the indices of the neighbor are valid, unchecked, and its value is within the specified range.

According to the arguments fed to the program, using these methods, the program reads the file and either grayscales the image (mode 1), or convolutes the image using a filter (mode 2), or quantizes the image with given range (mode 3), or simply writes the content into another file (mode 0).

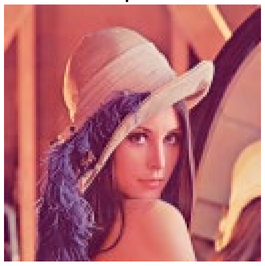
**Mahmut Alp Erkent**

**5.1.2020**

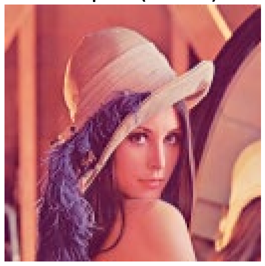
## 3) Output of the program:

Part 2:

Input

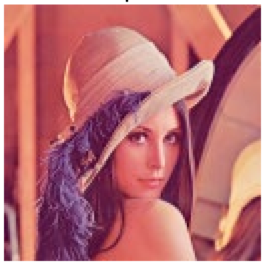


Output (same)

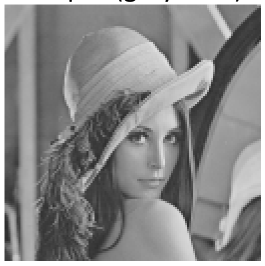


Part 3:

Input

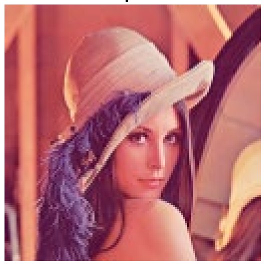


Output (grayscale)

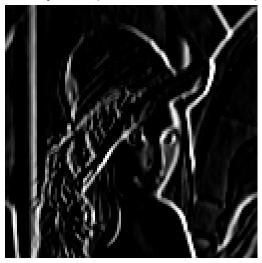


Part 4:

Input

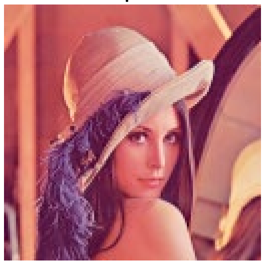


Output (vertical filter)

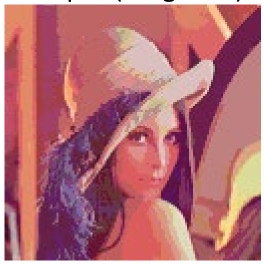


Part 5:

Input



Output (range: 25)



**Mahmut Alp Erkent**

**5.1.2020**

## 4) Conclusion:

With the code provided, I believe I have solved the problem correctly for each possible case. I tried to make my code as modular and as readable as possible.