University of Texas at El Paso - Department of Computer Science
CS1101: Introduction to Computer Science Lab
Spring 2019 Comprehensive Lab 2

Due Dates –

1. **Document containing pseudocode- April 4th, via Blackboard.**
2. **Java Program - April 8th (11:59 pm) via Blackboard**
3. **Demo to a member of the instruction team during labs or office hours no later than April 15th.**

1. Objective

This challenge lab will enable you to apply, in a practical scenario, the topics we have introduced in the class so far: input/output including files, variables, conditionals, loops, arrays, methods, and recursion.

2. Learning outcomes

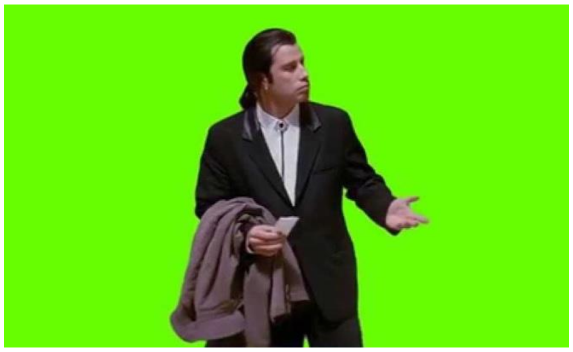
After completing this assignment, you will be able to:

- Analyze problems and express an algorithm solution using pseudocode.
- Implement a pseudocode algorithm in a high-level language, including the correct use of arithmetic and logical expression and simple input/output operations.
- Use the syntax and semantics of a high-level language to represent:
  - Basic variable types such as integer, real number, character, string, arrays
  - Assignment, arithmetic, and logical operations
  - Basic control structures: if-then, for-loop, while-loop
  - Arithmetic and logical expressions
  - Simple I/O operations
  - User-defined subprograms / methods

3. Your Challenge – Green Screen App

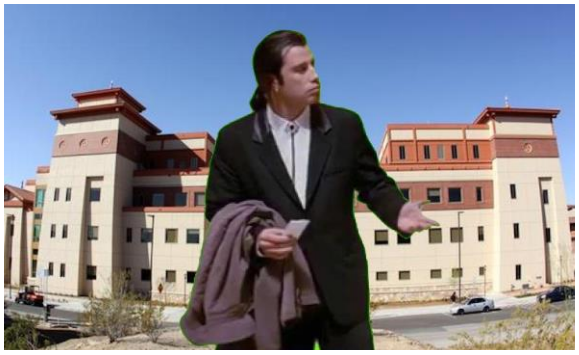
You have been hired by Sony Pictures to work on their next secret movie called “El Chuco Texas” starring Danny Trejo as the main character and directed by Quentin Tarantino. As part of your job, you are required to write an application called *AwesomeGreenScreenReplacer*. As the name suggests, your application will be used to process two images: a green screen picture and a background picture. Your code should replace all the green pixels found in the green screen picture with the corresponding pixels from the background picture. Here is an example of what your code needs to do:

Green Screen Picture:



Background Picture:

Output:



The following subsections describe the different tasks that your code should perform.

3.1. Get file names

In your main method or as a separate method (using an appropriate method name, e.g., `getFileName()`) get the name of the files you will use. By default, your code should use the following file names for the two input images: *greenScreenPic.jpg* and *backgroundPic.jpg*. However, the user must be given the option to provide different names if he/she wants to. The command line interface should look as follows

```
Use greenScreenPic.jpg as the file name of the green
screen picture? [Y/N]:

Use backgroundPic.jpg as the file name of the
background picture? [Y/N]:
```

If the user enters 'Y' as the answer, the default file name should be used. If the user enters 'N', your code should ask the user for the corresponding file name. If the user enters an invalid option, your code should display an informative message stating that the option they typed is invalid and proceed to ask them again.

3.2. Read image files

Your second task is to read the images by simply using the method called *readImage* found in *AwesomeGreenScreenReplacer.java* to do so. As you can see in the provided java file, *readImage* returns a 2D array of type Color. The 2D array represents the image you just read. Each element in this 2D array is a pixel. Pixels are represented with three values: red, green, and blue. You can read more about this here. The following code snipped shows you how to access the red, green, and blue values of the first pixel in a given image:

```
Color[][] image = readImage("greenScreenPic.jpg");

int pixelRow = 0;
int pixelCol = 0;

int redValue = image[pixelRow][pixelCol].getRed();
int greenValue = image[pixelRow][pixelCol].getGreen();
int blueValue = image[pixelRow][pixelCol].getBlue();
```

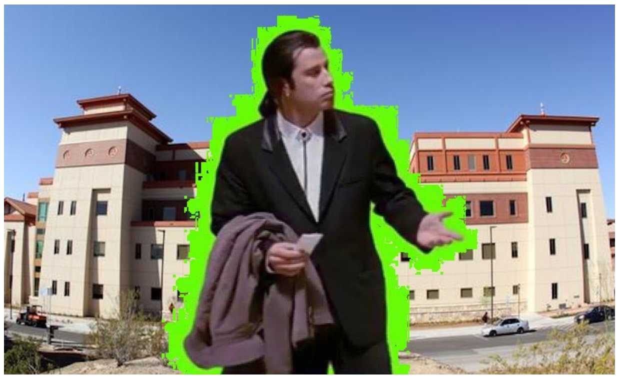
The values of *redValue, greenValue,* and *blueValue* range from 0 to 255 (inclusive). The "greener" a pixel is, the closer the value of *greenValue* will be to 255. Similarly, the "redder" a pixel is, the closer the value of *redValue* will be to 255. Naturally, the same applies to the blue channel.

3.3. Implement the method *isGreenPixel*

Your next task is to implement the method *isGreenPixel.* This method receives three numbers as input: *red, green,* and *blue.* The purpose of this method is to return true if and only if the pixel represented by red*, green,* and *blue* is a "green pixel" in the input image. The idea is to return true for all pixels that you want to replace in the green screen image. Your method should return false if the pixel that it gets as input is not to be replaced.

Hint 1: Before implementing this method. Randomly select some of the pixels from an area you think might be green from the John Travolta image and print the values of their *red, green,* and *blue* channels and try to see what makes a "green pixel." You should be able to see a pattern.

Hint 2: To the human eye, some pixels may appear green, this is not the case of a computer. Make sure to account for a range of different types of green so that your end-product doesn't look incomplete like the image shown to the right.



3.4. Implement the method *replaceGreenPixels*

Your next task is to implement the method *replaceGreenPixels.* This method receives two parameters: *greenScreenImg* and *backgroundImg*. These two parameters are of type Color[][]. They are 2D arrays, where each entry in the array represents one pixel. The purpose of this method is to create and return a new 2D array of type Color[][] where you will store the result of replacing the green pixels in *greenScreenImg* with the corresponding background pixels from *backgroundImg*. This method should call *isGreenPixel* to determine if a given pixel is green or not.

IMPORTANT: For your application to work, both images must have the same size. If you would like to test your program on your own images, you have to make sure they have the same dimensions. You can use this website to resize your images: https://resizeimage.net/

3.5. Display the resulting image

Once you have implemented *replaceGreenPixels,* you should call the *displayImage* method to display the resulting image. If what you see is not what you expected, go back to and make sure that the methods *isGreenPixel* and *replaceGreenPixels* are actually doing what they are supposed to. It's okay if the output image does not look perfect.

3.6. Ask the user if he/she wants to save the output image

Ask the user he/she wants to save the output image. If they do, ask them how they would like to name the image. Use the method *saveImage* to save the image to disk.

3.7. Write a recursive method to process multiple images

At this point, you should have a working application. To make it more interesting, let's add a recursive method. Make a copy of your working project before you proceed with the following task.

Let's give the user three options when selecting the background image, they want to use.

- Option A: Use the default name *backgroundPic.jpg*
- Option B: Let the user type the name of the background image
- Option C: Let the user type the name of a **folder** that contains multiple background images

If the user selects option C, you should perform the same process as before on all images stored inside of the folder the user specified (one at a time). You can use the method called *getFilesInFolder* to read the names of all the images stored in the specified folder. This method returns an array of strings, where each string represents the name of each of the files stored in the directory. Your task is to modify your code to add option C and run your logic on the given green screen image and all background images stored in the folder.

For example, let's assume you have a folder called "*BackgroundImages*" that contains all the images you want to "merge" with the green screen image. Let's assume there are three images inside of the folder. If you call *getFilesInFolder* and pass *"BackgroundImages"* as its parameter value, the method will return an array of size 3, where each entry corresponds to one of the images stored in the folder. Your code should proceed to run your *replaceGreenPixels* method three times; one for time for each background image in the folder.

To do this, you need to implement a **recursive** method. Do not use a for or while loop! You MUST use recursion to receive full credit. Half-credit will be given to non-recursive solutions. You might want to use this as the method signature for your recursive call:

```
public static void replaceGreenPixelsRecursive(Color[][]
greenScreenImage, String[] backgroundImagesPaths, int currentIndex) {. .
. }
```

The idea is that each time this method is called, *greenScreenImage* is processed with the background image at location *currentIndex.* The first time you call your recursive method, currentIndex should be the first index (0) or the last one, depending on the order in which you want to process the images.

4. Bonus

Think about a bonus feature you could include that will allow you to practice using input/output operations, a sequence of steps, conditionals or loops. You should be able to explain your bonus feature. For example:

- Allow the user to input the name of the images or read multiple background images.
- Additional testing cases – how can you check how robust is your system? Think about a scenario that you have not prepared your code for and how you could improve your code based on that scenario.
- Instructor Diego Aguirre has a green screen in his office. If you take pictures of yourself in front of that green screen and make your program work with those pictures, you will receive extra credit.

5. Task 1: Write pseudocode

Create a word document named “yourLastName-yourFirstName-ComprehensiveLab2.docx”. In this document, you will write describe your approach to the challenge.

5.1. *Task description*

In simple words, describe the task that was given to you and the tasks you will execute.

5.2. *Variables*

Define the variables and their corresponding datatype required to store the data needed by the *AwesomeGreenScreenReplacer* system. Make sure you use meaningful names. For each variable, include a sentence that describes how you’re going to use the variable, e.g., variable to store the balance of a checking account.

5.3. *Pseudocode*

Write the pseudocode to implement the functionality of *AwesomeGreenScreenReplacer*

An English description of your solution is enough for Comprehensive Lab 2. If you use the notation introduced in class, you will receive extra points for the pseudocode.

5.4. *Assumptions*

If something was not clear about your challenge and you make an assumption, please include it here.

6. Task 2: Implement *AwesomeGreenScreenReplacer* in Java.

You will implement the pseudocode created in your pseudocode document using Java. Your implementation should match your pseudocode. It is OK if after doing some implementation you refine your pseudocode.

6.1. *Download and annotate Java file.*

Download the AwesomeGreenScreenReplacer.java provided with this assignment through Blackboard. Modify the first lines of the code to include your name as follows:

```
/* CS1101 – Intro to Computer Science
Instructor: Aguirre OR Akbar OR Villanueva
Comprehensive Lab 2
By including my name below, I confirm that:
-  I am submitting my original work.
-  If I include code obtained from another source or I
   received help I am giving attribution to those
   sources as comments.
-  This submission does not incur in any academic
   dishonesty practice as described in the course
   syllabus.
Modified and submitted by: [YOUR NAME GOES HERE]
*/
```

6.2. *Modify the Java file.*

Using the pseudocode notation used in the lecture, write the pseudocode required to implement the functionality of *AwesomeGreenScreenReplacer*. Make sure you follow best practices when writing your code such as adding documentation when needed and using proper indentation.

6.3. *Run your program.*

Run your program and make sure that it's working as expected, e.g., by entering the inputs you used in your tracing example, you will see the expected outcomes.

7. Task 3: Submit your solution.

Using Blackboard submit the following:

- Your document named "yourLastName-yourFirstName-ComprehensiveLab2.docx".
- The modified **JAVA file** called AwesomeGreenScreenReplacer.java.

8. Grade percentage breakdown

15% - Pseudocode document (This includes the task description, variable names and description, pseudocode, and any assumptions)
5% - Appropriate use of input/output operations (in Java)
5% - Appropriate use of conditional (i.e., if-then) statements (in Java)
5% - Appropriate use of iterations (i.e., for-loop, while-loop) statements (in Java)
15% - Appropriate use of arrays (in Java)
15% - Appropriate use of methods (in Java)
10% - Appropriate use of recursive methods (in Java)
5% - Appropriate documentation (in Java)
5% - Appropriate notation and indentation (in Java)
15% - Program compiles, runs and contains the functionality required
5% - Student answers all questions during demo.
20% - Bonus feature

Penalization:

7.5% - Every 24 hours for up to 72 hours. For example, if you submit 36 hours later your maximum percentage is 85%.
10% - Not following the instructions for submission. Refer to the course syllabus for policies on academic dishonesty.

9. Tips

- Start early – today!
- Plan to work 3 to 5 additional hours outside the lab to complete this assignment.
- Ask clarifying questions to the instruction team if something is not clear.
- Plan to submit at least 1 hr. before the deadline to deal with potential Blackboard bugs.
- Keep it simple!