CS361 Assignment 3

Due: Friday, June 23, 2017 by midnight.

Don't forget that you'll be zipping your Java implementation files and README.txt file together and submitting a single zip file for the main program, and a separate zip file if you also do the extra credit. You should include either or both of the extra credit steps in one file. Don't forget to name your zip files according to the instructiions in General instructions.

Grading README file: Your zip file must contain a README.txt file. We're also using a program to grade your README.txt file. Any other names or formats of REAMDE file won't be accepted, e.g. readme.txt, readme, README, README.c, README.java. The README.txt file worths 3 (of the 10 points). If you use the incorrect REAMDE.txt file, you have to email the TA to handle this issue. Here is an example README.txt for this assignment 3. Note that this template is slightly different from assignment 1. Please download it and modify it directly. Don't use your own template!!!

For the README.txt file, the maximum point is 3.

1. First Five Lines [0.25 point] have to match the format that has been explained in General instructions..
2. Good description of your code [0.25 point]

   You have to write some sentences (at least 100 words) to explain high-level idea of program, some short description of the key/important function in your code.
3. You need to tell us how much you've finished [0.25 point]

   (1) Finished all requirements (2) Only finished parts of the requirements.

   If you're in the case (2), you have to list the unfinished things and write down the reasons for each, either no time or have some bugs. If grader thought you explanation is very good and detailed, you won't lose 0.5 point even if you didn't finish many requirements of program.
4. You need to answer 5 questions [2.5 point]

   0.5 point per question. See ``Hiding the Message'' section.

The grading scheme of this README.txt file has very little to do with the correctness of your program. Even if your program doesn't work well, please try your best to provide the details of the README.txt file, you still have a chance to get 3 out of 10 points.

Grading Program: For the program, the maximum point is 7.

1. You will get [2 points] if the program can be compiled. Before submission, make sure that your code can be compiled and executed on the Linux Machine (in our CS Lab). Even if your code is working on your own machine(Windows, ...), and not working on CS Lab machine, you won't get any points!
2. You will get [4 points] If the program can be run.
3. You will get [7 points] If the program is correct.

Background:

From Wikipedia: "Steganography is the art or practice of concealing a message, image, or file within another message, image, or file. The word steganography combines the Ancient Greek words steganos meaning 'covered, concealed, or protected', and graphein meaning 'writing'. The first recorded use of the term was in 1499 by Johannes Trithemius in his Steganographia, a treatise on cryptography and steganography, disguised as a book on magic. Generally, the hidden messages will appear to be (or be part of) something else: images, articles, shopping lists, or some other cover text. For example, the hidden message may be in invisible ink between the visible lines of a private letter."

In this assignment, you'll implement a simple steganographic algorithm by manipulating an image. In most modern image formats, there is ample "room" to hide a message. For example, the bmp format uses three bytes for each pixel in RGB format. Each byte indicates the intensity of the corresponding color (Red, Green, Blue). The following is an example of an image in this format: UT Picture. Your task is to take an image and hide a message in it. In practice, the resulting picture should not be visibly different from the original. You should also be able to recover the message from the modified image. Ideally, your program should be able to work on various image formats: bmp, png, jpeg. (I believe that the java library utilities for reading BufferedImages allow reading of multiple image formats, but I wouldn't swear to that.)

Deliverables:

You will be submitting your code electronically. Make sure that your code is well-annotated (commented) and follows good coding style. Clearly identify the members of your team (one or two people). Include a README file explaining how to run your program. Also, in your README file, address the issues I raise at the end of this file.

The primary file name should be `Steganography.java`. Students are welcome to organize their assignments into multiple files and must submit those files as well. However, the main method must be in `Steganography.java` and the TAs should be able to compile your program with `javac *.java`.

The Assignment

Write code to perform the following steps:

1. Read an image file. Some sample code to show how you might do this is here: Sample Java Code.
2. Print statistics on the image: filename, number of pixels in the file, image height, and image width. (see the sample code)
3. Input another file containing an ascii text message and generate a new image that is visually identical to the original image, but contains the hidden message. If the original was in a file named "xxxx.ext" the new file should be called "xxxx-steg.ext" That is, preserve the file extension, if any.
4. Your code also should be able to extract the message from a modified image. Specify an output file name on the command line.

Names of both the image file and message file should be passed on the command line. Additionally, a flag should be used to indicate whether you are encoding or decoding. For example, consider the following commands.

```
Steganography -E image.png my-message
Steganography -D image-steg.png my-message-out
```

The first indicates encoding the contents of the ascii file `my-message` into the image in `image.png`. This should produce an image file named `image-steg.png`. The second line takes the modified image and extracts the hidden message. It puts it into a file called `my-message-out`.

Hiding the Message:

Each pixel in the image is represented by 24 bits of information. You can extract each of the RGB values from the pixel and store one bit of information in each. For example, suppose the pixel has RGB values of (126, 240, 122), and the next three bits you'd like to store are 1, 1, 0. The encoding scheme is as follows: If the bit you want to store is 0, make the corresponding byte even; if the bit is 1, make the byte odd. For example, the RGB triple above might store (1, 1, 0) by becoming (127, 241, 122). (On average, only half of the RGB values will change need to change, further reducing the visible effects.) Reading the bits is merely reading the even/odd-ness of successive RGB values. When you do change the parity, it doesn't seem important whether you raise or lower the value. But be careful not to decrement 0 or increment 255, as these may cause an error or make a much more visible change to the pixel. Notice also that, given this scheme, it may not be possible to recover the original image exactly from the modified image.

Using this approach you can store the message in the image using 3 bits per pixel. Store a 0 byte to indicate the end of the message. If your input message is too long to store in the given image, you should store as much as you can. But be sure to allow enough room to store the final 0 byte; also print an error message to the terminal indicating that you've truncated the message. (Recall that you've already computed the number of pixels, so know how many bits of message the image can store.) I came up with this method off the top of my head. I don't see any reason why it shouldn't work; but please let me know if you do.

In your README file, you should address the following:

1. Comparing your original and modified images carefully, can you detect *any* difference visually (that is, in the appearance of the image)?
2. Can you think of other ways you might hide the message in image files (or in other types of files)?
3. Can you invent ways to increase the bandwidth of the channel?
4. Suppose you were tasked to build an "image firewall" that would block images containing hidden messages. Could you do it? How might you approach this problem?
5. Does this fit our definition of a covert channel? Explain your answer.