Assignment 2: Ada Programming (25%)

BASIC IMAGE PROCESSING

An image is a simple matrix of pixels, i.e. a 2-dimensional or rectangular array of pixels. PGM is short for Portable Gray Map, which is a means of storing an image in a file. If an image is to be stored in formats like JPEG, or TIFF there needs to be quite a bit of work done to encode the image data. PGM which supports *grayscale images* is one of a family of different file formats: PPM supports colour images, and PBM supports binary images.

All these file formats have very simple headers and support both ASCII text or binary format. A PGM file consists of two parts: the header and the data. A PGM header is typically composed of a "magic" identifier, which signifies the type of file, two integers representing the width and height of the image in pixels, and an integer representing the maximum grayscale value. An example of a PGM file is shown below:

```
P2
24 7
255
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  3  3  3  3  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0 15  0
0  3  3  3  0  0  0  7  7  7  0  0  0 11 11 11  0  0  0 15 15 15 15  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0  0  0
0  3  0  0  0  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

The magic identifier is P2, which indicates an ASCII file (P5 is the binary form). The size of the image is 24 columns (width) by 7 rows (height), and the maximum value is 255, signifying that all "pixels" in the image have values between 0 and 255.

An image in this context is an 8-bit grayscale image, i.e. each pixel has a value between 0 and 255. An image is normally stored by a programming language as an array of integers (the smaller the datatype the better).

TASK

Write a program in Ada to perform some image processing operations on a grayscale image stored in an ASCII P2 PGM format.

1. Create an Ada package, `imagePGM` to deal with the input and output of images stored using the PGM "P2" file format. The two subprograms contained in the package should be

`readPGM()` and `writePGM()`. Make sure to provide relevant error checking, e.g. the subprogram `readPGM()` should generate an error if the wrong magic identifier is present, or if there are any other inconsistencies with the input file.

1.1. `readPGM()` should take as input a filename and return a record representing the image.
1.2. `writePGM()` should take as input a record representing an image, and write the image to file as a P2 PGM format.

2. Create an Ada package, `imagePROCESS`, which includes a series of image processing algorithms, implemented as subprograms (see below for details).Your package should include a `record` structure to hold the image (array and dimensions).

3. Create a main "wrapper" program, `image.adb`, which allows the user to interact with the two packages above, reading in images, manipulating them, and writing them to file.
   3.1. It should include a subprogram `getFilename()` which gets the username for the file to be read or written from the user. For reading, an error should be generated if the file does not exist. For writing, an error should be generated if the file does exist, and request whether it should be overwritten. An example call might be:
   `fn = getFilename("r")` to obtain the filename of an image to read.

Image Processing

There are numerous resources on the Internet that deal with basic image processing. The simplest image enhancement techniques are those which apply a mathematical transform to each single pixel in an image, often known as *point operations*. If an image is represented by `I`, then `I(x,y)` represents the pixel at row `x`, and column `y`.

For example the original image `I`, is transformed using function `F`, to create a new image `Z`. The values `x` and `y` represent the `x` and `y` coordinates of the pixels in the 2D array, i.e. I_{xy} is the pixel at row `x` and column `y`.

$$Z_{xy} = F(I_{xy})$$

The list of image processing algorithms to incorporate into the package `imagePROCESS` include:

- A subprogram `imageINV(I)` which performs an image inversion:

  $$F(I_{xy}) = abs(255-I_{xy})$$

- A subprogram `imageLOG(I)` which performs a logarithmic transformation (mapping a narrow range of low intensity pixels into a wider range of output levels):

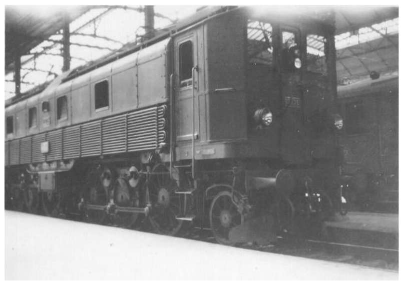
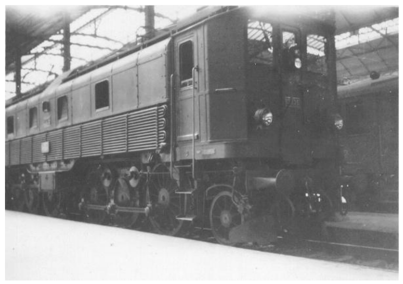
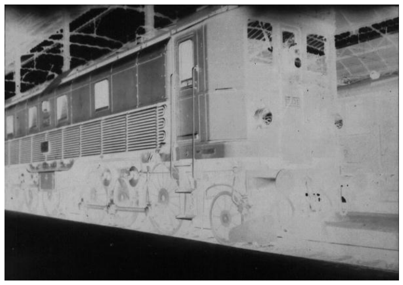
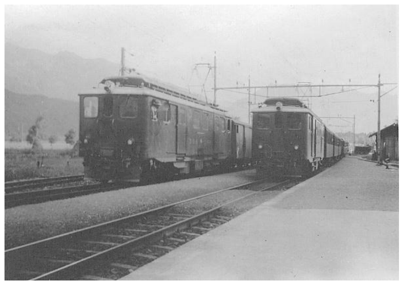
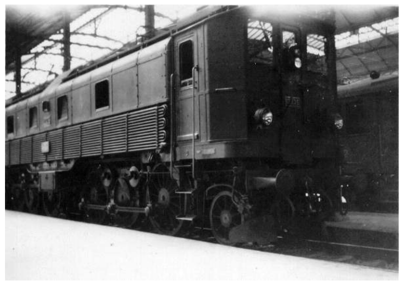
  $$F(I_{xy}) = log(I_{xy}+1)$$

- A subprogram `imageSTRETCH(I,imin,imax)` which performs a contrast-stretching. A contrast stretching transformation which is a process that expands the range of intensity levels in an image. The equation for contrast stretching on a grayscale image is:

$$F(I_{xy}) = 255\frac{I_{xy} - I_{min}}{I_{max} - I_{min}}$$

- I_{min} is the minimum intensity value present in the image, and I_{max} is the maximum intensity value present in the image

Here are some examples (the top row are the original images, the bottom row the processed images):

Image inversion	Log transform	Contrast stretching
		
		I_{min}=64, I_{max}=251
		

EXTRA TASK

By completely the above task correctly, you have the potential to earn up to **90%** of the grade for this assignment. By going beyond the material given and completely the extra section, you have the potential of earning the remaining 10%.

HISTOGRAM EQUALIZATION
(See Appendix for algorithm details)

In the package `imagePROCESS` :

1. Add a function `makeHIST(I)` which calculates a histogram of an image `I`, returning the histogram as a 1D array.

2. Add a function `histEQUAL(I)` which performs the histogram equalization of an image `I`, returning the enhanced image.

COMPILING

Please do not include a Makefile, and make sure your program compiles in the following manner:

```
> gnatmake -Wall imagePGM.adb imagePROCESS.adb image.adb
```

SAMPLE OUTPUT

As shown in the description.

SKILLS

- Ada programming, re-engineering by translation, program comprehension.

ASSIGNMENT INFORMATION

REFLECTION REPORT

Discuss your program in a **one (1) page (single-spaced)** reflection report, explaining the decisions you made in the process of reengineering your program. It should provide a synopsis of your experience with Ada.

Some of the questions that should be answered include:

- Was Ada well suited to solving the problem?
- What particular structures made Ada a good choice?
- Benefits / limitations?

DELIVERABLES

The submission should consist of the following items:

- The reflection report (PDF).
- The code (well documented and styled appropriately of course).
  `image.adb imagePGM.adb imagePGM.ads imagePROCESS.ads imagePROCESS.adb`
- Both the code and the reflection report can be submitted as a ZIP, TAR, or GZIP file.

STYLING & COMMENTS

Style consists of mnemonic variable names, indentation, and the use of whitespaces and paragraphing. The purpose of good style is to make the meaning of your program clear to someone who has never seen it before, cannot run it, and cannot talk with you. Documentation consists of in-code documentation. Examples of qualities to look for include:

- Are variable names well chosen?
- Are comments relevant rather than simple repetitions of the code?
- Do comments point out key sections of code, indicate special cases, or make assertions?
- Are the indents 3 or 4 spaces? Do not use tabs or 2 space indenting (please check "convert tabs to spaces" in your editor)
- Is whitespacing used to separate parts of the program to provide clarity?

SKILLS

- Ada programming, program comprehension, problem solving

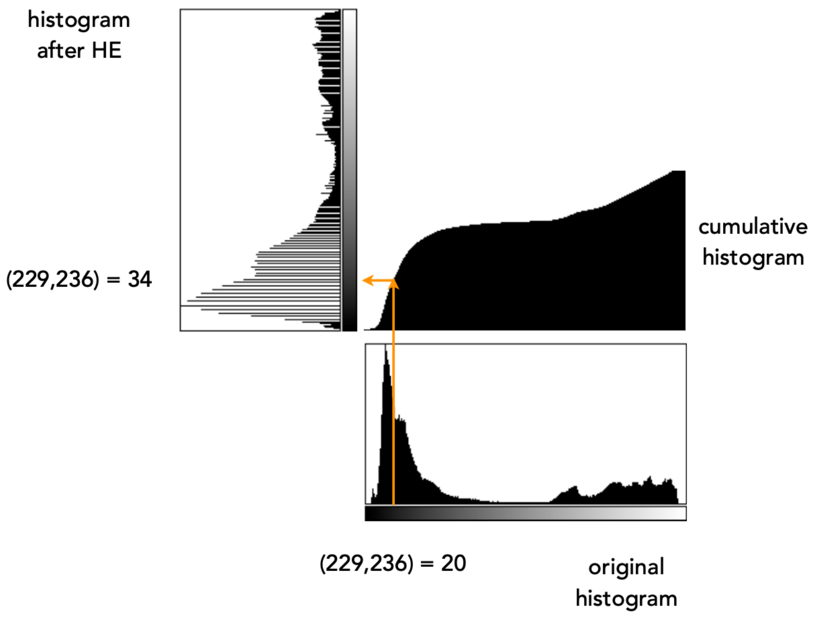
APPENDIX: EXTRA TASK

HISTOGRAM EQUALIZATION

Histogram equalization (HE) is a popular technique for improving the appearance of an image. Its function is similar to that of histogram stretching. Histogram equalization is a technique where the histogram of the resultant image is a *flat* as possible. It is based on the probability distribution of the grayscale values in the image. The process consists of a series of steps:

1. Calculate the histogram of the grayscale image. The histogram of an 8-bit image is a 256 element array, with each element (or bin) representing the number of times a particular grayscale appears in the image. For example H[256] is the histogram, then if the value of H[1] is 197, then it means that 197 pixels in the image have the grayscale value 0.
2. Calculate the probability density function (PDF) of the histogram by dividing each bin in the histogram by the total number of pixels in the image.
3. Calculate the cumulative histogram (CH). This is a 256-element array where each value is the cumulative value from the PDF. For example CH[1] = PDF[1], CH[2] = CH[1] + PDF[2], CH[3] = CH[2] + PDF[3] etc.
4. Multiply the CH by 255 and round.
5. Map the new grayscale values from the results of Step (4) using a one-to-one correspondence.

The illustration below shows the mapping of a pixel at location [229,236] in an image using the histogram equalization algorithm. The initial value of the pixel is 20, however using the mapping function calculated, the new value is 34.


histogram
after HE
(229,236) = 34
cumulative
histogram
(229,236) = 20
original
histogram