# FIT3042 System Tools and Programming Languages

## Semester 1, 2014

## Assignment 2

**Some Basic Image Processing Using Perl**

**Worth: 20% of final mark**

**Must be completed individually**

**Hurdle: 40% of assignment marks across assignments 1 and 2**

**Due: Friday, 30th May, 2014, 11.55pm**

One of the main things for which Perl is used is for taking data in a format that cannot be read by a program and converting the data into a format which can be. This happens a lot in eScience where there may be some mature, complex, large legacy codes. Nobody feels confident about changing these codes. They would rather change the data to suit the legacy code than to change the legacy code to accept a different form of input data.

The largest source of data is probably image data. In the earlier days of computer image processing there were a great number of image formats. The Monash University Image Processing Library (MULIPS) was a library of C programs that enabled people to manipulate images. At the time when MULIPS was first written, there was a great range of image formats and MULIPS was written to process a number of image formats which where specific to the Monash environment. The names for these images ended in extensions like `.bmg`, `.img`, `.smg` and `.cif`.

A greyscale image is an image whose pixel values are small, non-negative integers. A commonly used convention is that 0 represents black and the most positive integer value represents white. Intermediate values represent shades of grey. Often a single byte is used to represent the grey scale value of a pixel. In this case, black is represented by `0`, white by `255` and shades of grey by values between `0` and `255`.

### Task 1: a Perl program for creating to `img` files - 5 marks

A `pgm` file consists of two components: a header and the image contents. The header consists of lines of text which are terminated by `\n`. The first two characters must be the 'magic number', `P2` for a `pgm` file with the pixel values in ASCII, or `P5` for a `pgm` file in binary (one or two bytes per pixel). The P2 format is rarely encountered these days so we will be referring to the P5 or 'raw' format whenever we refer to `pgm` in the rest of this document.

After the magic number, the header then contains 3 integer values as ASCII - the number of columns, the number of rows and the maximum pixel value. The header can contain comment lines, line which begin with '#'. After the header come the actual pixel values. Pixel values are stored one row at a time. For a `pgm` in raw form, if we assume that the image data does not contain any comment lines, once we know the number of rows and columns, the image data can be extracted by using a Unix tool like `tail` to extract the last (number of rows) x (number of row bytes). This technique assumes that the file only contains one image.

More information on the `pgm` can be found on the final page of this assignment as well as by looking at Wikipedia for information on `Netpbm`.

Write a Perl program, `pgm2img`, which is given an image file in `pgm` format. The program should read the header, and write the header information to standard output. If the file is in raw form, i.e. it has a magic number of `P5`, `pgm2img` should cause a Unix program like `tail` to write the image pixel values to a file. You can assume that the `pgm` only contains a single image.

The name for the file should be the base name for the image (the part preceding the `pgm` extension, followed by an underscore, the number of rows, an underscore, the number of columns and finally the extension `img`. For example, if the file `lennagrey.pgm` contains a 512 x 512 greyscale image in raw format, then the resulting output file name should be `lennagrey_512_512.img`.

**Task 2: a Perl `img` to `pgm` Converter - 5 marks**

The `img` format is a simple format for storing greyscale images. It uses one byte to store the value of a pixel. Thus, the pixel values are unsigned integers that range from `0` to `255` inclusive. An `img` file contains just the actual pixel values, with one byte per pixel. The pixel values are stored by rows.

There is no information in the body of a `img` file to say how long each row or column of the image is. The name of the file is used to indicate how many rows and columns an image has. The format for the file name for a `img` file is as follows:

```
    filename_nrows_ncols.img
```

`nrows` gives the number of rows in the image and `ncols` gives the number of columns. Thus, if a file has the filename `lennagrey_512_512.img`, then it has 512 rows and 512 columns. If the file has the name `television_480_640.img`, then it has 480 rows and 640 columns. This means that each row has 640 pixels.

Write a Perl program, `img2pgm`, that accepts a filename via the command line. The program checks that the file name is in the correct format for the name of a image stored in `img` format and gives an error message if it is not. If the file is in the correct format it then reads the file and produces a file in `pgm` format.

If you find Perl routines that allow you to write out a file in `pgm` format, you may use these in completing this task, but if you do use third party code, be sure to indicate this in your documentation.

Note, in debugging your code you can use `pgm2img` from Task 1 to create sample data.
Hint: when debugging, you may want to create very small test images, e.g. 4 x 4 or 8 x 8.

### Task 3: a Perl `smg` to `pgm` Converter - 5 marks

Files in `smg` format are used to store greyscale images where the pixel values take up two bytes (the `s` in `smg` stands for `short`). The `smg` file format is like the `img` file format in that the body of the file contains the raw values of pixels, stored by rows, while the filename gives information about the number of rows and columns in the image. For example, the file name

```
    filename_nrows_ncols.smg
```

indicates that the file has pixels whose values are small integers that take up 2 bytes of storage and has `nrows` with `ncols` pixels in each row.

A complication with storing integers as two bytes is that the order in which the bytes are stored may vary depending on the hardware. If 16 bit integer is stored in 2 bytes, we can term the byte which holds the least significant bits of the integer value the `lo` byte and the byte which holds the more significant bits, the `hi` byte. In the `smg` file format there is no provision for indicating the appropriate byte order. An integer value may be stored `lo` byte then `hi` byte or it could be stored `hi` byte then `lo` byte.

Write a program called `smg2pgm` that is given the name of a file in `smg` format. In addition, after the name of the file on the command line, there may be an optional argument which is either `lohi` or `hilo`. This argument represents the byte order, with `lohi` meaning that the `lo` byte of a 2 byte integer value is stored before the `hi` byte and `hilo` indicating the opposite byte ordering. If no byte ordering argument is present, the program should assume the order is `hilo`.

The program should produce a file which has the same basename as the `smg` file, but have an extension of `.pgm` and be in `pgm` format and, by default, contains exactly the same pixel values.

`smg2pgm` should have a second optional command line parameter, `one_byte`. If this is present, the output image should use only a single byte to store the value of a pixel. To map 2 byte integers to one byte integers, the program should take the maximum pixel value in the `smg` file and map it to 255 and map the lowest pixel value to 0. This mapping should be linear. For example, the value exactly half way between the lowest and the highest integer values in the `smg` file should be mapped to either 127 or 128.

### Task 4: A Perl program to threshold an image in either `img` or `smg` format - 5 marks

Thresholding an image means taking an image file and mapping pixel values to either white or black, depending on whether the pixel value is strictly less than the threshold value or greater than or equal to the threshold value. Often thresholding involves computing an image intensity histogram. For each possible distinct pixel value there is a corresponding entry in the histogram which counts how often pixels with that value occur in the image.

Write a Perl program, `threshold2pgm` that is given the name of a file which contains an image in either `img` or `smg` format and produces a thresholded image in `pgm` format. The pixels in the thresholded image have the value 0 for black, if their pixel value is below the threshold and the value 255 if their threshold value is above the threshold.

The Perl program will be given a parameter on the command line, `threshold_value` where `threshold_value` will be an integer in the range 1 to 99 inclusive. `threshold2pgm` will interpret `threshold_value` as 'threshold the image so at least `threshold_value` percent of the pixels in the thresholded image are black. The threshold it chooses should be the smallest possible integer value so that at least `threshold_value` percent of the pixels are mapped to black.

If `threshold2pgm` is given a filename of the form

`ImageFileName_nrows_ncols.img` or `ImageFileName_nrows_ncols.smg`

it should produce an output file containing the threshold image with the name with the corresponding name

`ImageFileName_threshold.pgm`

For example, invoking `threshold2pgm` with

`threshold2pgm lennagrey_512_512.img 40` should produce a `pgm file` where at least 40% of the pixels are black and the rest are white. The file should be named

`lennagrey_40.pgm`

`threshold2pgm` should only make one pass through the input file and during that pass it build up a histogram of the pixel values. [A histogram will have, for every possible pixel value, a count of how often that pixel occurred.]

**Bonus Marks - 2 marks**

In Task 3, the user needs to specify the correct byte order for storing the 2 bytes of a 16 bit integer. Perhaps the program can come up with an 'educated' guess.

A characteristic of grey-scale images is that the value of a pixel is often highly correlated with the values of its neigbouring pixels. For example, for all pixels which have a neighbour which precedes them on their row, we can compute a correlation coefficient value for the value of that pixel and the value of the preceding pixel. For most grey scale images this value is often close to 1.0. This suggests that we can try forming the pixel values in the `hilo` order and computing the correlation coefficient and we can also compute the correlation coefficient for pixel values assumed to be in the `lohi` byte order. Whichever byte order leads to the larger correlation coefficient we can declare to be the correct byte order for this `smg` image.

Amend your program for Task 3 so that if it is not given a parameter indicating which byte order to try, it tries determining the byte order automatically and use that byte order in the image conversion. It should output that byte order to standard output. How well does this heuristic work? [You can find information on computing correlation coefficients from many sources, including Wikipedia ...]

**Testing and error handling**

Your software should produce correct output for all legitimate input files, and fail gracefully, with useful error messages, when faced with incorrect user inputs (for instance, the wrong number of command-line arguments). It is your responsibility to test this! You will be penalized if your program crashes, hangs, or otherwise misbehaves, even on malformed user inputs.

**Target system**

Your program will be compiled and run on the same Ubuntu virtual machine configuration as used in labs. It is *your* responsibility to make sure the program compiles and runs on this machine.

If you wish to use a Perl module from CPAN, you need to ask on the Assignment 2 Moodle forum.

**Packaging**

You must submit a single archive file in gzipped `tar` format through Moodle. See `info tar` for information how to create a gzipped tar archive.

You should include a README file, in text format, that describes (at a minimum):
your name and student ID number, how to compile and run the program, what functionality is and is not supported, and any known bugs or limitations.

You should also complete the electronic plagiarism statement as normal.

**Good programming practice**

Perl can be a "write-only" language if you are not careful!

Things you should be looking for (because the markers will) include the following:

- Appropriate use of subroutines and modules (if you choose to use modules).
- Appropriate function and variable names
- Appropriate variable scope.
- Error handling! If something can fail, check for it and handle appropriately. Where not specified, you can choose whether to terminate execution or continue as you judge best, but silently failing or silently doing the wrong thing is cause for losing marks.
- Code that compiles! Code that generates a compile-time error (the Perl executable compiles scripts to bytecode before running) will receive a failing grade.

**Marking criteria**

Your assignment will be marked on:

- Functional correctness: does the system do what it is supposed to do?
- Efficiency: are your programs efficient in their use of resources, primarily CPU time and memory?
- Design: are your programs cleanly engineered?
- Coding practices, including readability.
- Compliance with submission instructions.

*Due date*: Friday, May 30th, 2014.

The following information about the `pgm` file format is extracted from
`http://netpbm.sourceforge.net/doc/pgm.html`

THE FORMAT

The format definition is as follows. A PGM file consists of a sequence of one or more PGM images. There are no data, delimiters, or padding before, after, or between images. Each PGM image consists of the following:

(1) A "magic number" for identifying the file type. A pgm image's magic number is the two characters "P5".

(2) Whitespace (blanks, TABs, CRs, LFs).

(3) A width, formatted as ASCII characters in decimal.

(4) Whitespace.

(5) A height, again in ASCII decimal.

(6) Whitespace.

(7) The maximum gray value (Maxval), again in ASCII decimal. Must be less than 65536, and more than zero.

(8) A single whitespace character (usually a newline).

(9) A raster of Height rows, in order from top to bottom. Each row consists of Width gray values, in order from left to right. Each gray value is a number from 0 through Maxval, with 0 being black and Maxval being white. Each gray value is represented in pure binary by either 1 or 2 bytes. If the Maxval is less than 256, it is 1 byte. Otherwise, it is 2 bytes. The most significant byte is first.

    A row of an image is horizontal. A column is vertical. The pixels in the image are square and contiguous.

    All characters referred to herein are encoded in ASCII. "newline" refers to the character known in ASCII as Line Feed or LF. A "white space" character is space, CR, LF, TAB, VT, or FF (I.e. what the ANSI standard C isspace() function calls white space).