PPM Image Transformations

Learning Objectives

Upon completion of this assignment, you should be able:

1. To develop, compile, run and test C programs in a Linux environment
2. To navigate Linux command lines reliably

The mechanisms you will practice using include:

- Linux command lines: manual pages, Linux commands, input/output redirection
- C Programming: structs, pointers, memory allocation, getopt

Program Specification

NAME

`ppmcvt` – convert ppm files

SYNOPSIS

`ppmcvt` [bg:i:r:smt:n:o:] [file]

DESCRIPTION

`ppmcvt` manipulates input Portable Pixel Map (PPM) files and outputs a new image based on its given options. Only one option that specifies a transformation can be used at a time.

In the synopsis, options followed by a ':' expect a subsequent parameter. The options are:

-b
: convert input file to a Portable Bitmap (PBM) file. (DEFAULT)

-g:
: convert input file to a Portable Gray Map (PGM) file using the specified max grayscale pixel value [1-65535].

-i:
: isolate the specified RGB channel. Valid channels are "red", "green", or "blue".

-r:
: remove the specified RGB channel. Valid channels are "red", "green", or "blue".

-s
: apply a sepia transformation

-m
: vertically mirror the first half of the image to the second half

-t:
: reduce the input image to a thumbnail based on the given scaling factor [1-8].

-n:

tile thumbnails of the input image based on the given scaling factor [1-8].

-o:
write output image to the specified file. Existent output files will be overwritten.

EXIT STATUS

`ppmcvt` exits 0 on success and 1 on failure.

EXAMPLES

`ppmcvt`
read input PPM file from standard input and write converted PBM file to stdout

`ppmcvt -g -o out.pgm in.ppm`
convert the PPM image in.ppm to a PGM image in out.pgm

`ppmcvt -s in.ppm`
apply a sepia transformation to the PPM image in in.ppm and output the new image to stdout

`ppmcvt -n 4 -o out.ppm in.ppm`
tile 4 1:4-scaled (quarter-sized) thumbnails of the image in in.ppm into a new PPPM image in out.ppm.

ERRORS

`ppmcvt` should print to the standard error output stream exactly the specified line and then exit under the following circumstances:

"Usage: ppmcvt [-bgirsmtno] [FILE]\n": malformed command line

"Error: invalid channel specification: (%s); should be 'red', 'green' or 'blue'\n”

"Error: Invalid max grayscale pixel value: %s\n"

(File errors are handled by the provided pbm library.)

Implementation Details

Image File Formats

PPM, PGM and PBM files are simple (and inefficient) ASCII text file image formats comprising a small header followed by integer values that represent each pixel in the image. Wikipedia has a good description here: https://en.wikipedia.org/wiki/Netpbm.

Image Transformations

Ignoring whitespaces, your program should produce *exactly the same output images* as mine. My program uses floating point arithmetic for all intermediate calculations then converts the resulting floats to integers as appropriate.

Bitmap:

To compute black and white bits from RGB pixels use:

$$Average(R+G+B) < PPMMax/2$$

Grayscale:

To compute grayscale pixels from RGB pixels use:

$$\frac{Average(R+G+B)}{PPMMax} \times PGMMax$$

Sepia:

For the sepia transformation, compute RGB pixels as follows:

$$NewR = 0.393(OldR) + 0.769(OldG) + 0.189\,x(OldB)$$
$$NewG = 0.349(OldR) + 0.686(OldG) + 0.168\,x(OldB)$$
$$NewR = 0.272(OldR) + 0.534(OldG) + 0.131\,x(OldB)$$

Mirror:

Vertically reflect the left half of the image onto the right half.

Thumbnail:

The output thumbnail should be 1/n the size of the original file, where n is the input scale factor. Shrink the input image simply by outputting every n^{th} pixel in both dimensions starting with the first.

Nup:

Tile n 1/n scale thumbnails, where n is the input scale factor. The output image should be the same size of the input image.

Requirements and Constraints

This assignment aims to make you familiar with some 'C' programming basics. As such, we impose several requirements and constraints on your implementation:

1. You must use getopt() to process your program's command line inputs.
2. You must use the provided pbm library (described below)
3. You may use only the following library or helper functions:
    a. C Memory Allocation: malloc(), realloc(), calloc(), free()
    b. Command line parsing: getopt()
    c. C string functions: strlen(), strcmp()
    d. Other: strtol(), exit()
    e. pbm library
4. Intermediate storage: You must use dynamically allocated memory to store any intermediary image data. That is, you may not create temporary image files nor use static arrays (for example, `int image[MAXHEIGHT][MAXWIDTH]`). Instead, you should create an array like: `int **image` and dynamically allocate the precise memory needed depending on the image size.
5. You must free any dynamically allocated memory immediately when it becomes no longer useful.

Submission

FOLLOW THESE INSTRUCTIONS PRECISELY

Requisite files:

- Sources: all the .c and .h *that you implemented* to build your program
- Make file: a make file named, `Makefile`, that builds the `ppmcvt` program.
    - Your make file should assume your .c and .h files are in the current working directory
    - Your make file should build the `ppmcvt` in the current working directory
- README: you may submit an optional README file with comments, feedback, specifying known issues or problems, etc.

You should create a subdirectory of your CS551 directory named lab0 and place these files there. Be sure to a make clean, followed by a new make to test your solution. Thats what we will do.

PBM.H

Our provided pbm library (.h and .c files) does the following:

1. Defines structs for PBM, PGM and PPM image types
2. Defines I/O routines to read or write images to or from a PBM, PGM or PPM file. (Note: The read routine does not handle image files with embedded comments.)
3. Declares memory allocation/deallocation routines for PBM, PGM and PPM structs.

You must not modify pbm.h nor pbm.c: you will not submit these files; we will build your program using our original versions.

For your convenience, the contents of the .h file are:

```
typedef struct {
   unsigned int ** pixmap; //h x w (2-dimensional) bitmap array
   unsigned int height, width;
} PBMImage;

typedef struct {
   unsigned int ** pixmap; //h x w (2-dimensional) pixel array
   unsigned int height, width, max;
} PGMImage;

typedef struct {
   unsigned int ** pixmap[3]; //Three h x w (2-dimensional) pixel arrays, one array
for R, G and B values, respectively.
   unsigned int height, width, max;
} PPMImage;

/** YOU MUST IMPLEMENT THE FOLLOWING NEW/DEL FUNCTIONS **/

//new functions return a properly initialized image struct of the appropriate type, with
all necessary memory for the pixel array, pixmap, properly malloc'd
PPMImage * new_ppmimage( unsigned int width, unsigned int height, unsigned
int max);
PGMImage * new_pgmimage( unsigned int width, unsigned int height, unsigned
int max);
PBMImage * new_pbmimage( unsigned int width, unsigned int height );

//del routines free ALL memory associated with image struct including the input
image struct pointer
void del_ppmimage( PPMImage * );
void del_pgmimage( PGMImage * );
void del_pbmimage( PBMImage * );

/** THESE FOLLOWING FUNCTIONS ARE IMPLEMENTED FOR YOU IN pmb.c **/
PPMImage * read_ppmfile( const char * filename );
void write_pbmfile( PBMImage *image, const char * filename );
void write_pgmfile( PGMImage *image, const char * filename );
void write_ppmfile( PPMImage *image, const char * filename );
```