Programming Concepts

Exercise 4

November 26, 2015

FH JOANNEUM
University of Applied Sciences

Institute of Internet Technologies & Applications

Introduction

In this exercise you will have the chance to apply all your newly gained knowledge about the C language. You will write a simple command line based tool to read and manipulate *NetPBM* files.

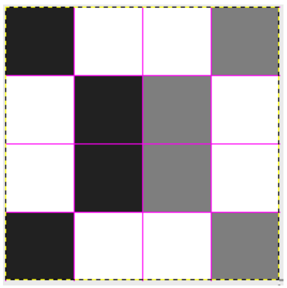
0.1 Teams

Since the final exercise is a little bit more involved you are allowed to do this exercise in teams of two. *Larger groups than two are not permissible!* I would suggest that you use the practice of pair programming to solve the problem instead of trying to split the exercise.

NetPBM Format

The NetPBM is a collection of simple bitmap image formats. The file extension (depending on the type) are typically `.pbm`, `.pgm` and `.ppm`. Regardless of the actual image type NetPBM images are always stored uncompressed and contain a simple plain text header. We will take a closer look at the format with the aid of concrete examples[1]

```
P5
# CREATOR: GIMP PNM Filter Version 1.1
4 4
255
!~!~!~!~
```



`simple_sample.pgm`

On the right we see a zoomed version of the image since the original image is very small (4x4) pixels

Header

The first line specifies the type of NetPBM image. The following headers a known:

Magic Number	Format	Image Encoding
P1	.pbm (black/white) 1 bit color depth	ASCII
P2	.pgm (gray scale) 8 bit color depth	ASCII
P3	.ppm (RGB color) 24 bit color depth	ASCII
P4	.pbm (black/white) 1 bit color depth	Binary
P5	**.pgm (gray scale) 8 bit color depth**	**Binary**
P6	.ppm (RGB color) 24 bit color depth	Binary

*For this exercise you need only to implement the **P5** format!* All other formats can be rejected by your program.

[1] You will find more examples in your assignment

The header is followed by an arbitrary amount of comment lines. (lines starting with # and ending with a new line character)

The next header line (the first line that does not start with a #) specifies the dimension of the image (width and height). The next line specifies the maximal intensity of a single pixel. For this image format it should always be `255`. This is also the last header entry in the next line the image content will begin.

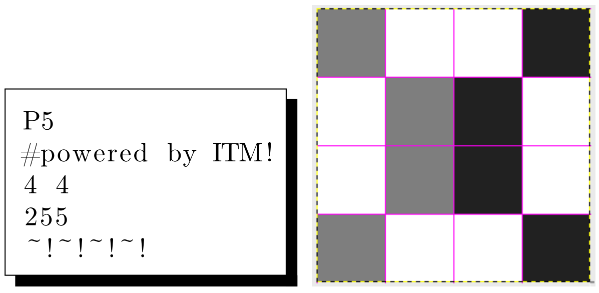
Image Content

Depending on the concrete image type the image data can either be encoded in ASCII or binary. Since only the `P5` format is of interest to us only the binary encoding will be elaborated.

In `P5` we are dealing with 8 bit gray scale images, which means that every pixel is represented by a byte. The value 0 represents a black pixel and the value 255 a white, value between are various shades of gray. The first byte of the image data represents the left upper pixel and last byte represents the right bottom one. From the information in the header we know exactly how many bytes of image data are expected in the image.

Implementation

The program should read an image file into memory, flip the image and write the flipped image back to the hard disk. The image should be flipped that the top left pixel becomes the bottom right pixel and so on. This means that the image is flipped in a 180° angle. Below you will find again the same example file but this time flipped.



```
P5
#powered by ITM!
4 4
255
~!~!~!~!
```

`simple_sample_flipped.pgm`

Skeleton Code

In the assignment you will find two header files (`img.h` and `flip.h`) which contain function prototypes, constants and structs. If you choose you can use these file as a base for you program. You normally do not need the modify the given header files. The implementation of function prototypes should be in the respective C file i.e. `img.c` and `flip.c`. The command line parsing and the entry point (main function) of your program should occur in `main.c`

If you are not satisfied with the overall design imposed by the given header files, feel free to make your own implementation from scratch. However make sure that you should use at least the following features: **structs**, **dynamic allocations** and **multiple source** files. Also make sure that your own implementation takes the same command line arguments

Concrete implementation

Your program should accept two command line arguments. The first parameter specifies to *location* to the PGM file that you want to flip. The second parameter specify the *destination* where the flipped image will be saved to. If the destination file already exist it should simply be overwritten! The execution of the program will involve of the following steps:[2]

- The input file should be opened `argv[1]`
- The content of the image file should be read an the `PbmImage` structure should be created by means of the `pbm_image_load_from_stream` function.
- The image data contained in the previously loaded `PbmImage` structure should be flipped by using the `pbm_image_flip` function.
- The now flipped `PbmImage` structure should now be written back to the hard disk by the `pbm_image_write_to_stream` function.

All these outlined step require of course proper error handling. Additionally make sure that all resources that are allocated (heap memory, file handles) are properly free'd at the end of the program. Make sure that your program has no improper memory access and and does not cause/depend on undefined behavior. [3]

Additional requirements

You can use all function provided to you by the C Standard library however **third party libraries are not allowed!**. Your program should adhere to the C99 standard an must compile with the `-Wall` flag without any warnings. If debug output is present in your program it should only visible by defining a debug proprocessor symbol such as `-DDebug`

Helpful standard library functions

The following sections list a few standard library methods which might be helpful for you implementation. These are only suggestions and you are free to use any standard library function you want. For an overview of the standard library or an more detailed explanation of the below mention functions use a reference site such as `http://www.cppreference.com`

- `strcmp`: Checks strings for equality.
- `fgets`: Reads from a file until a newline is encountered.
- `fscanf/sscanf`: Reads data from a file or strings and extracts values given by the format string
- `fread`: Reads binary data from a file into memory.

[2]If you do not use the given skeleton you can ignore the part about function names and structures
[3]Hint: Use the tool `valgrind` to check for these issues

- `fwrite`: Writes data from memory into a file.
- `feof`: Checks if the end of a stream has been reached.

Submission

Submit the exercise on the E-Learning platform. Since this exercise will have multiple source files **you should zip** your exercises. Upload your assignment to the exercise folder with the name **Qualification Exercise**. Please use the following naming convention: `exercise_04_<lastname>_<lastname>.zip`

When you zip your solution take care that your source files are in the 'root' of your zip archive and not in a sub folder.