UNIVERSITÄT DUISBURG ESSEN

NES Networked Embedded Systems

Programming in C/C++

Exercise Sheet 03
!!! Deadline: Tue, Nov 19th, 23:55 !!!

Preparation:
In order to complete the following tasks, you need to have libjpeg and the corresponding headers installed on your system.

- On Linux systems you need to install 'libjpeg-dev' (Debian based distros) or 'libjpeg-devel' (Red Hat based distros).
- On OS X you install libjpeg via Homebrew or MacPorts.
- On Windows you can install libjpeg by using the package manager of Cygwin or MinGW.

You can also compile the library from source (by just including it in your submission).

Task 01:
In the following Tasks you will write a program that reads a JPEG file into memory, resizes it to a desired size using a bilinear filter, and then compresses it using the JPEG format. Images are stored as a giant matrix of pixels. Each pixel has information about its color. JPEG usually uses the 24-bit RGB color space. So each pixel stores the amount of red, green and blue color in 8-Bit with integer values between 0 and 255. This leads to 16,777,216 different colors for a single pixel.

Write a struct to store a single 24-bit RGB Pixel in memory. In the template file there is an empty struct `pixel_rgb_t`. Use this struct and fill in the missing fields.

Download the provided main.c file template and complete it by implementing the following functions

ABGESCHLOSSEN

i. Write this struct as memory efficient as possible. Ideally your struct should allocate only 24 bits.

ii. Afterwards implement the conversion from a JPEG Scanline to your data structure. Go into the `load_jpeg` function. Comment on what you are doing.

iii. Do the same for the `save_jpeg` function but this time you have to write a function to convert your `pixel_rgb_t` to a JPEG scanline. A scanline is a complete row of the image. It has the following format:

red|*green*|*blue*|*red*|*green*|*blue*|*red*…

Comment on what you are doing.

iv. Write the main function. The program should accept and input filename and an output filename. (e.g. **imagecopy -i myfile.jpg -o myoutfile.jpg**). To test your code, you should load the file, read it into memory, and then save it to the output file without modification. The output image should look exactly like the input image.

Hint: You have to link against libjpeg. Pass –ljpeg to GCC in your Makefile. Make sure to add the linker option at the end of the compile command or it might not work.

Hint2: Checkout getopt. It's an easy way to parse commandline parameters.
Submission: Your executable should be called **imagecopy**.
Points: (code 25pts, comments 10pts)

Task 02:

Please make sure you implemented Task 01. Otherwise you can't complete this task.

Extend Task 01 to also accept the new width and the new height as commandline arguments. Use the flags –w and –h for this purpose. Use the struct `image_size_t` to store these values.

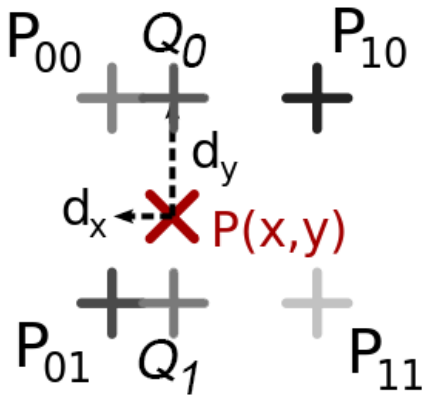
Don't forget to check if the passed values are valid. Print an error message and exit with code 1 if a value is invalid. Comment on what values are invalid and how you detect them.

You don't need to check the input and output files. They are already checked by the load and save functions.

This task is a preparation for Task 03, where the resizing function will be implemented. You should still save the image with the original size.

Hint: You have to link against libjpeg. Pass –ljpeg to GCC in your Makefile.
Submission: Your executable should be named **imageresize**
Points: (code 10pts, comments 5pts)

Task 03:
Please make sure you implemented Task 01 and Task 02. Otherwise you can't complete this task.
Now we are going to implement the actual resizing function `resize_image` using bilinear interpolation. In the following we will explain how bilinear interpolation works:



The Point P(x,y) is the new point you want to calculate. First, you calculate the 4 nearest pixels in the source image (P00, P10, P01, P11).
dx and dy are the differences between the new point, P, and the points in the original image in the x and y directions respectively.

When you scale from 1200x800 to 600x400 then both dx and dy are 0.5. When you scale to 600x600 dx is 0.5 and dy is 0,75.

Afterwards, you compute two values Q0 and Q1. Q0 is between P00 and P10 and Q1 is between P01 and P11.

They are computed using the following formula:

$$\boldsymbol{Q_0 = (1 - d_x) * P_{00} + d_x * P_{10}}$$

$$\boldsymbol{Q_1 = (1 - d_x) * P_{01} + d_x * P_{11}}$$

To compute P(x,y) use the following formula:
$\boldsymbol{P(x,y) = (1 - d_y) * Q_0 + d_y * Q_1}$

These formulas should be used per color channel. So compute the red, the green and the blue channel using these formulas. You can write new functions if you want to.

Refactor your main method to do the following:

i. Parse the command line arguments
ii. Load a jpeg file into memory.
iii. Resize the image to the desired size
iv. Write the resized image to the output file.

Test your function by resizing any JPEG image you want. If you implemented this algorithm correctly, you should be able to scale up and down.

Comment on your implementation.

Note: Depending on the output image size, the efficiency of your code and the power of your processor, the resizing process can take a few seconds to complete.

Hint: You have to link against libjpeg. Pass –ljpeg to GCC in your Makefile.
Submission: Your executable should be named **bilinear.** Don't include any images.
Points: (code 45pts, comments 5pts)

Task 04 - Bonus:
Currently the resizing does not keep the input image's aspect ratio.

Change the main method in such a way that you can pass only the height or the width to you program. The program should then compute the missing variable (height or width) based on the aspect ratio of the input image.

Note: You have to read the JPEG file into memory to know the size of the source file.

Hint: You have to link against libjpeg. Pass –ljpeg to GCC in your Makefile as the last option (otherwise it might not work).
Submission: Your executable should be called **proportional**
Points: (code 10pts, comment 5pts)