Materials

Course materials for the laboratory are in the directory `/home/share/informatyka_3` on the server `info3.meil.pw.edu.pl`. Before you start work, copy them to your home directory. To do this, execute the command

```
cp -r /home/share/informatyka_3/. ~
```

In your home directory two directories will appear, `drop` and `japan`, containing images.

If you want to copy files to your local computer, please use the command to download files from a remote server described previously.

```
scp -r studxy@info3.meil.pw.edu.pl:/home/share/informatyka_3/. ~
```

If you don't have the access to the server, you can download the images directly: info3.zip

Image processing

`convert` program

The basic program that we will use in class is the `convert` program from the ImageMagick library. This program is used to convert and change image properties. He can also add elements to the image and even create images from scratch. It's easiest to see how to use it by analyzing examples.

ATTENTION: Before you start working, copy pictures to a temporary directory!

Perform the following operations and check the effects.

- `convert file.gif file.jpg` - converts a file in the `GIF` format to the `JPG` format,
- `convert file1.jpg -resize 50% file2.jpg` - reduces the image twice,
- `convert file1.jpg -resize 100 file2.jpg` - reduces the image so that the shorter dimension is 100 pixels,
- `convert file1.jpg -resize 100x100 file2.jpg` - reduces the image to fit in a square of 100 by 100 pixels,
- `convert file1.jpg -resize 100x100\! file2.jpg` - reduces the image to exactly 100 by 100 pixels,
- command

```
convert -size 320x85 canvas:none -pointsize 72 -fill red \
-draw "text 20, 55 'Magick'" magick.jpg
```

- will create the image `magick.jpg`, with the text "Magick" marked on it (the characters "spaces" and "slash" at the end of the line mean that the command will be continued in the next line).

Exercises

Write a script that:

- Resizes all `.jpg` files contained in the current directory.
- Resizes all `.jpg` files contained in the current directory and places them in another directory.
- It will convert all `.jpg` files to `.gif`, adding the extension (`file.jpg` -> `file.jpg.gif`).
- It will convert all `.jpg` files to `.gif`, changing the extension (`file.jpg` -> `file.gif`).
- It will put text on each photo, using the argument

```
-pointsize size -draw "text x, y 'Text'"
```

- It will put a frame on each photo (argument `-border 20x20`) with the current date (command `date`).
- The date of creation of this image will be marked on each photo (it can be extracted using the command `stat -c% y file`).
- Resizes all images in the "drop" directory and joins them into animation with the command `convert *.jpg animation.gif`.

Automation

Try to write scripts that perform the following tasks:

- Adds a frame to the image and writes a selected `EXIF` information to it (only photos in the `japan` directory have EXIF information). This information can be extracted using the `identify -format"%[EXIF:*]" file.jpg` command. This can be, for example, the model of the camera.
- Writes numbers from 0 to 10 to the screen using a loop.
- Prints a number from 0 to the number given as the script argument to the screen.
- Combines all images in a given catalog, reduced to 10 by 10 pixels, into one large `JPG` image. The `-append` {.bash} argument joins the images vertically and the `+append` argument horizontally. For example, the command `convert 1.jpg 2.jpg 3.jpg +append 4.jpg` will combine three images horizontally into one.
- In an analogous manner, it will combine images from the `drop` catalog into one large image. Images should be combined to form a 10-by-10 checkerboard. Hint: combine the images `drop-00*.jpg` horizontally, then `drop-01*.jpg`, etc. Then combine all these longitudinal pictures vertically.
- Decomposes the `GIF` animation into individual `JPG` images. He will add text to each of the received images and then put them together in a new `GIF` animation. The script should delete temporary files (i.e. single images obtained by unfolding the original animation).

Sometimes writing a script can be tedious. This applies especially to situations where the script must be extended and at the same time will be used only once. For example, the following script generates an image composed of three overlapping images: a black and white background, and two files `1.jpg` and `2.jpg`.

```
#!/bin/bash

convert -size 90x60 canvas:white \
-size 90x30 gradient: -append -rotate 90 \
\( 1.jpg -resize 90x90\! -clone 0 -compose CopyOpacity \
+matte -composite -repage +60+0 \) \
\( 2.jpg -resize 90x90\! -clone 0 -compose CopyOpacity \
+matte -composite -repage +120+0 \) \
-compose Over -mosaic overlap_series.jpg
```

Analyze the example above and write a script that will create an image of all files contained in the specified directory. To this end, the script should:

- generate a script `tmp.sh` containing the command analogous to the above, only for a larger number of images (use `echo line> tmp` {.bash} for the first line and `echo next line` {.bash} >> tmp for each next)
- grant execute permissions for the script `tmp.sh`,
- run it.

Loading binary data

The `convert` program can load binary data by interpreting it as an image. For example, it could be an array containing "char" type values. Write a program `picture.c` containing the following code:

```
#include <stdio.h>
#include <stdlib.h>

#define N 100
#define M 100

int main() {
  int i;
  unsigned char *tab;

  tab = malloc(N * M * sizeof(unsigned char));

  for (i = 0; i < N * M; i++) {
    tab[i] = 255 * i / (N * M - 1);
  }

  fwrite(tab, sizeof(unsigned char), N * M, stdout);
  free(tab);

  return 0;
}
```

Then compile it using the command

```
gcc picture.c -o picture.out
```

and do it

```
./picture.out > picture
```

We now have a binary file containing char (1 byte) numbers from 0 to 255. We can convert it to the `JPG` image by entering:

```
convert -size 100x100 -depth 8 gray:picture picture.jpg
```

Note: Instead of the "char" type, we could use a "float" array and numbers from $ [0, 1] $. Then, however, we would have to convert the image using the command:

```
convert -size 100x100 -depth 32 \
-define quantum:format=floating-point gray:picture picture.jpg
```

Exercises

- Reduce any photo to a size (exactly) 100 by 100 pixels.
- Modify the created command to convert this image to a binary file.
- Based on the `picture.c` program, write the program `filter.c`, which:
  - reads an array from the standard input (the "fread" function is used for this),

    ```
    size_t fread(void *ptr, size_t size, size_t count, FILE *stream);
    ```

    where: `*ptr` is the pointer to the array, `size` is the size of the array element, `count` is the number of elements to load, and `* stream` is the pointer to the stream on which the operation is performed (in our case it will be `stdin`)
  - for each number read, `x` will find a value that will allow it to invert the corresponding color (note that for numbers of type `char`, black is 0 and white is 255),
  - the transformation result will be sent to the standard output.
- Try to pass the selected image through such a "filter" and check the result. Remember that we can send information to the standard program input using `<`.
- Write a script that will reduce all files in the current directory to 100 by 100 pixels, convert to binary files, pass through a filter and save the results as `JPG` files.