Programming 2 (SS 2023)
Saarland University
Faculty MI
Compiler Design Lab

Project 2

Prof. Dr. Sebastian Hack
Pascal Lauer, M. Sc.
Marcel Ullrich, B. Sc.

The name of this project is inspired by the open source project *convert*[1], a command line tool for image manipulation. As the original *convert* project contains very extensive work, the version you have to implement only covers a fraction of *convert's* features. The selected tasks are 1) reading and writing images, 2) simple image manipulation e.g. rotating an image and 3) performing a flood fill operation you may know as the paint bucket tool in image editing software like Photoshop, GIMP or Microsoft Paint. Please refer to the respective sections for detailed descriptions. In the following, we will describe the technical setup.

To obtain the project use

```
git clone ssh://git@git.prog2.de:2222/project2/$NAME $FOLDER
```

where `$NAME` specifies your CMS username and `$FOLDER` is the folder the project should be placed in.
You can edit the files in the `src` folder using any text editor. We recommend using Visual Studio Code as it is already installed within in the virtual machine. To compile the project use

```
make
```

in the root project of the repository. After that you can use the following command to run your program:

```
./bin/convert command [options] > output.ppm < input.ppm
```

- Substitute `command` by the name and arguments for desired functionality. (Described in each section.)
- Substitute `input.ppm` by the file path you choose as input.
- Substitute `output.ppm` by the file path you choose as output.
- You can find sample input files in `test/data`.
- Use `./convert_opt` instead of `./convert` for the compiler-optimized version.

Please note that you should not substitute <, >. By default the executable reads from and writes to the command line. By using `> output.ppm` the output is redirected to `output.ppm`. By using `< input.ppm` the input is redirected to `input.ppm`. Use the command

```
./tests/run_tests.py
```

to run the local tests for your project. Make sure to first build your project using `make`. `make check` will build the project and run all local tests. You can use the `-f` flag to run a specific test. For example,

```
./tests/run_tests.py -f "public.read_and_write.basi0g01"
```

to run the test `public.read_and_write.basi0g01`. Existing parameters (flags) are documented at relevant places in this document.

The project is structured in a way that each section corresponds to one header (.h) file. Each functionality described in the tasks needs to be implemented in a separate function, declared in these files. Additionally, there are a few pre-existing structs and one pre-implemented function that should be taken into consideration before starting the implementation of the tasks. These can be found in the `structures.h` and `file_io.h` files respectively.

Your project will be compiled with sanitizers that, among other things, check for correct memory handling and undefined behavior. They give you more helpful error messages in case of crashes, but will also lead to failed tests if your memory management is faulty. Make sure to read the notes in section 5 about the test system.
You have two weeks to complete this project, meaning that we evaluate the last: `git push` until 22.05.2023, 23:59, MESZ.

Happy Coding :)

[1] `https://imagemagick.org/script/convert.php`

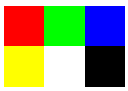
1 Reading and Writing Image Files (3 Points)

Your first task is to implement file input and output functionality for the ASCII variant of the portable pixmap format (P3) with the maximum value for each color always set to 255. The format is very well described on the Wikipedia website.[2] To summarize what you need to know about the format:

1. The P3 PPM format is designed for storing RGB bitmaps.
2. The file ending is `ppm`.
3. The format of the P3 PPM file is as follows:
   - The file always starts with the "magic number" identifying the format, which in our case is `P3`.
   - It follows with two numbers specifying width and height of the image.
   - Next, the maximum value for each color is specified. In our case, this must always be set to `255`.
   - Finally, each pixel of the image is described by three numbers between 0 and 255 specifying the RGB color code of the pixel, and there must be specified all pixels of the image in the row-major format. That is, if the width is W and the height is H, then there must be $W \times H$ triples of numbers, each triple describing a pixel, and the first W triples describe the first row of the image, the next W triples describe the second row, and so on.
   - The whitespaces between the elements are meaningless, i.e., you can use spaces, tabs, or newline separators as you wish.
   - You can completely disregard comments. Test images do not contain any comments.
4. The following is an example of a PPM file with 3 columns and 2 rows (i.e., width is 3 and height is 2):

```
P3
3 2
255
255   0   0   0 255   0   0   0 255
255 255   0 255 255 255   0   0   0
```

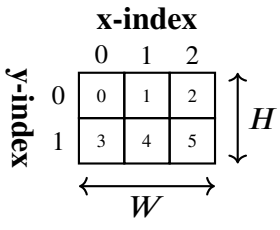
   It represents the following image:



The two main structures used for representing images in the C program can be found in the `structures.h` header file. The structure `pixel_t` holds the RGB value of a pixel, and the structure `image_t` holds the whole image:

```
struct image {
  int w;
  int h;
  pixel_t *img;
};
typedef struct image image_t;
```

The member `.w` stores the width of the image (i.e., the number of columns), and `.h` holds the height of the image (i.e., the number of rows). The array `.img` contains all pixels of the image in a row-major format, i.e., the pixel in the column x and row y (both indexed from 0) is stored in the array `.img` at index $(y \times \texttt{.w}) + x$. The indices in the example image are the following:



[2] `https://en.wikipedia.org/wiki/Netpbm`

To implement the read/write functionality take a look a the `src/file_io.*` files. The function to read an image is called `image_read`. The function to write an image is called `image_write`. Both functions take a FILE pointer as argument that is used as input for reading and output for writing. When reading, the input file must match the correct format, or else the function should return -1. Please note that you should reserve space for the image data. E.g. using `malloc`. To read and write the contents to or from a file, it is recommend to utilize the functions `printf` and `scanf` respectively.

To test your implementation, you can run the command

```
./bin/convert read < input.ppm
```

to test reading the input file. And you can run the command

```
./bin/convert read-and-write > output.ppm < input.ppm
```

to test reading and subsequent writing of the same image.

2 Simple Image Manipulation (4 Points)

For the next task take a look at the functions located in `src/image_edit.c`. Here you implement simple image manipulation techniques. More concretely you will implement to rotate, mirror and resize images. You will be able to perform simple local tests for this tasks without the read/write functions from section 1 using the GUI described in section 4.

2.1 Mirror and Rotate

To rotate an input image by 90 degrees clockwise or counterclockwise, the user must implement the functions `rotate_counterclockwise` and `rotate_clockwise` respectively. To mirror an input image horizontally or vertically, the user must implement the functions `mirror_horizontal` and `mirror_vertical`. These functions are easiest to understand visually. Please refer to figure 1 for a clearer understanding of the expected output.

`original image`

`rotate-clockwise`

`rotate-counterclockwise`

`mirror-horizontal`

`mirror-vertical`

1: Rotating and mirroring of `test/data/owl2.ppm`.

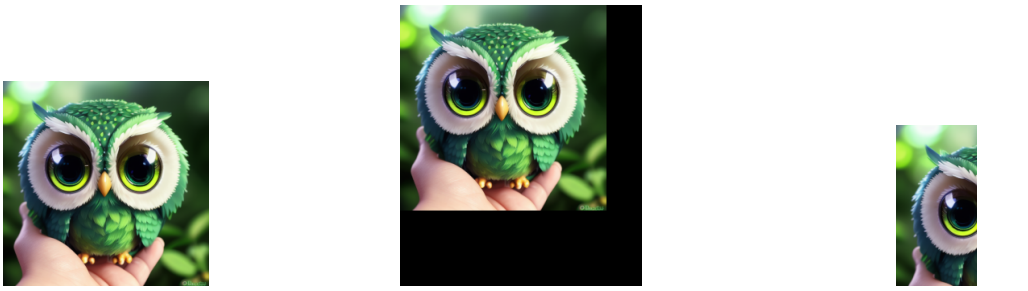
To test your program use the following commands, corresponding to the functionality used to transform the example picture with the respective annotation.

```
./bin/convert mirror-horizontal > output.ppm < input.ppm
./bin/convert mirror-vertical > output.ppm < input.ppm
./bin/convert rotate-clockwise > output.ppm < input.ppm
./bin/convert rotate-counterclockwise > output.ppm < input.ppm
```

2.2 Resize

To resize images, the user must implement the `resize` function in `src/image_edit.c`. This function takes an image, desired width and desired height as arguments and modifies the image so that it has the specified dimensions. If the target width or height is smaller than the width or height of the input image, the image gets cropped. If the target width or height is larger than the input width or height, the image is extended with black pixels (RGB (0, 0, 0)). Please note that this means that the new image needs more memory than you previously allocated. Figure 2 provides an example of this functionality. The implementation can be tested with the following command:

```
./bin/convert resize WIDTH HEIGHT > output.ppm < input.ppm
```



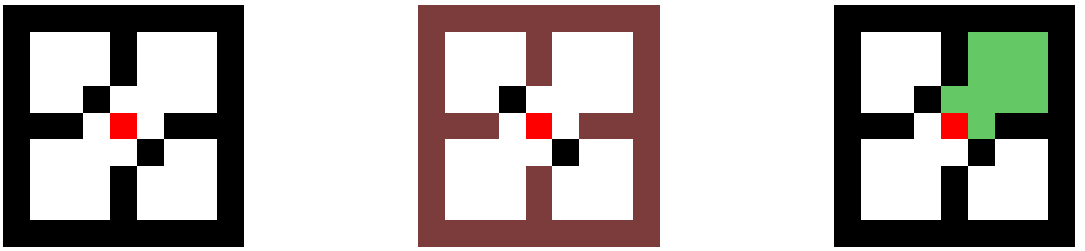
`original (256x256)` `resize 300 300` `resize 100 200`

2: The sample image transformed by the resize operation.

3 Flood-Fill Algorithm (4 Points)

Your last task is to implement the 4-directional version of the flood-fill algorithm in the file `src/flood_fill.c`. That is, given an image, a position of a pixel at column X and row Y, and a target RGB color (R, G, B), the function `flood` modifies the input image so that the pixel at (X, Y) and all horizontally or vertically connected pixels with exactly the same color as (X, Y) are re-painted to the specified color (R, G, B). If (X, Y) points outside the image, nothing changes.
We again believe that this operation is best understood visually, and Figure 3 provides examples of the expected output with respective commands. However, if you are interested in further details, you can find them on the Wikipedia page for flood-fill [3]. You can test your implementation with the command:

```
./bin/convert flood X Y R G B  > output.ppm < input.ppm
```



`original (9x9)` `flood 0 0 125 60 60` `flood 7 1 100 200 100`

3: Examples of the flood-fill operation.

[3] `https://en.wikipedia.org/wiki/Flood_fill`

4 Graphical User Interface

Testing your implementation locally via the command line requires you to implement the read and write functions from section 1. If you have not yet implemented the read and write file functionality and want to test your image manipulation functions, you can use the provided Graphical User Interface (GUI). To start the GUI, you can use the command:

```
./bin/convert gui
```

In the GUI, you will find a button on the lefthand side for each functionality that you need to implement. Click on the corresponding button to test your implementation with the pre-loaded sample image. The GUI allows you to visually inspect the results of your implementation, which can be helpful for debugging. Please note that the GUI will be closed, and an error will be printed to the command line, if you try to use functionality that you have not implemented yet.

5 Evaluation

As in the previous project, for every task there are public, daily as well as eval tests. You have to pass all public tests corresponding to a section to obtain the points for it. The daily tests are not available locally. These are scheduled to run after you upload your project to our server (git push). The eval tests will be used to rate the projects upon completion. These tests are unavailable to you.
The test system will only use `.c` files from the `src` subfolder of your project that were provided with the project. In particular, new files as well as changes to headers (`.h` files), Makefile and the test scripts will be ignored. Make sure that your code can be compiled under these conditions to avoid the failure of all tests. Also, make sure that the existing command line parameters are not modified.

In this project, you will obtain points based on the successful implementation of subtasks according to to the following table:

Section	**Points**	**Public Tests**
read-and-write	3	public.read_and_write.basi0g01, public.read_and_write.owl, public.read_and_write.small1, public.read_and_write.small2, public._.imgbroken1, public._.imgbroken2
image-manipulation	4	public.rotate_counterclockwise.* public.rotate_clockwise.* public.mirror_horizontal.* public.mirror_vertical.* public.resize.*
flood	4	public.flood.*

Important: A test is only considered as successful if the output is correct *and* there are no memory leaks detected by the address sanitizer.

The tasks are all common programming problems. It is perfectly reasonable to look up further information on the internet. If you copy any parts of your program from the internet, you need to cite that, otherwise it counts as plagiarism.
It may be helpful to add your own tests to the test framework included in your project repository. You can find more information about this in the `run_tests.py` script in your project repository.
To generate ASCII `.ppm` images according to the specification above from `.png` or `.jpg` images, you can use `https://neuralcoder3.github.io/ppm-converter/`.