Project Two: My Image Processor

Out: June 2th, 2022; Date: 11:59pm, June 16th, 2022

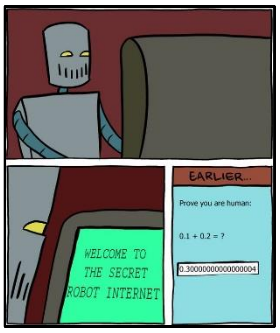
I. Motivation

The goal of this project is to help you practice basic C++ programming and in particular: program arguments, I/O stream, array manipulations, and function pointers. It also gives you an idea about how to organize and write more modular and reusable code. Accessorily, you will also learn some basics about image processing and gain some more knowledge about Linux commands.

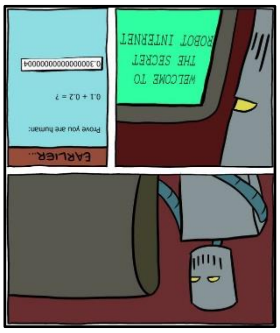
II. Introduction

In this project, you will implement a Linux command, called `mip` (for my image processor), that offers some simple image processing functionalities. This command reads an image from a file, applies an image transformation to it, and writes the resulting image into another file. As an illustrative example, the following command flips the image in input.ppm upside down and writes the results in the file named output.ppm:

```
./mip -i input.ppm -o output.ppm -t vertical_flip
```



input.ppm

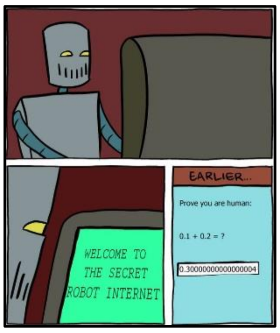


output.ppm

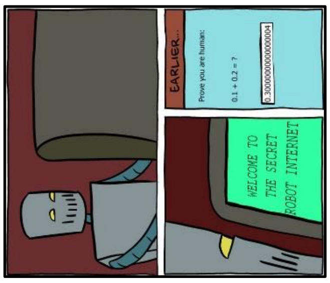
See below for a presentation of the ppm file extension.

If no input or output file is provided, the command will read from the standard input or output respectively. With such behavior, it is then possible to run this command consecutively and apply several image transformations. For instance, the following command flips the input image upside down and rotates it:

```
./mip -i input.ppm -t vertical_flip | ./mip -o output.ppm -t rotate90
```



input.ppm



output.ppm

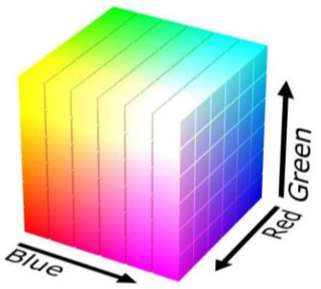
Recall "**|**" is called a pipe. It allows to redirect the output of the command that appears in its left to the input of the command that appears in its right. **Note** that one call of the program only applies one transformation.

In the next sections, we provide a short introduction to image processing, explain your programming assignments, describe the expected behaviors of your `mip` command, and finish with the usual explanation about submission and grading.

III. Image Processing

An image can be thought of as a 2D matrix. The dimension of the image (i.e., width and height) corresponds to the size of the matrix. One component of this matrix corresponds to one point of this image, which is called a **pixel** (picture element). A colored image is usually represented in an RGB (red, green, blue) format. In that case, a pixel corresponds to a vector of three elements representing the intensities of those three colors.

In mathematical notations, an RGB image is an element $\boldsymbol{I}$ of $\mathbb{R}^{w \times h \times 3}$ where w is the width of the image and h is its height. We will denote $\boldsymbol{I}_{i,j}$ its RGB vector in $\mathbb{R}^3$ at pixel position (i, j). We will follow the usual convention in programming of indexing from 0. Therefore, $0 \leq i < w$ and $0 \leq j < h$.



RGB Color Space

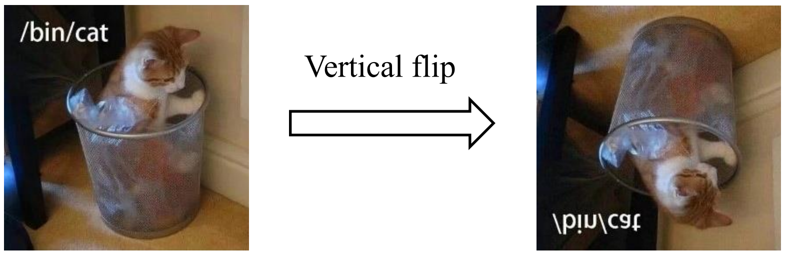
In a computer, each component of the RGB vector is usually represented by a non-negative integer bounded by M. In this project, we will assume that they can be represented by a char and can therefore only take values between 0 and $M = 255$ (inclusive).

Part I. Operations on Images

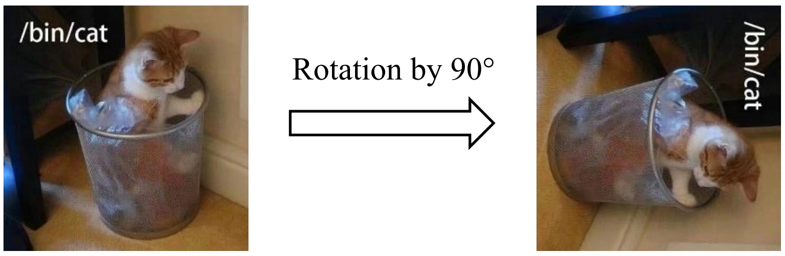
Notations: $\boldsymbol{I}$ in $\mathbb{R}^{w\times h\times 3}$ denotes the input image and $\boldsymbol{J}$ denotes the resulting image after an operation is applied.

Various operations can be performed on images, as you may know if you have already used any image processing applications (e.g., gimp or photoshop). In this project, we will consider the following simple operations:

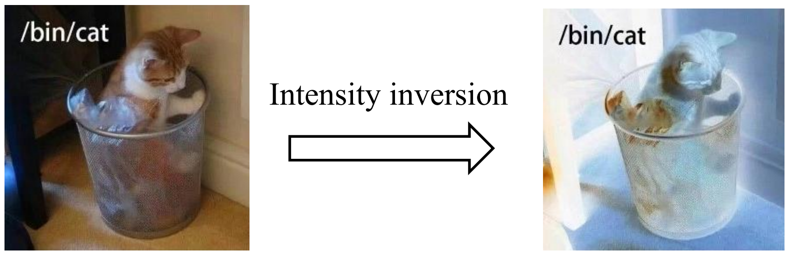
- ***Vertical flip***: After applying this operation on an image $\boldsymbol{I}$, the resulting image $\boldsymbol{J}$ in $\mathbb{R}^{\boldsymbol{w}\times\boldsymbol{h}\times\boldsymbol{3}}$ is such that $\boldsymbol{J}_{\boldsymbol{i},\boldsymbol{j}} = \boldsymbol{I}_{\boldsymbol{i},\boldsymbol{h}-\boldsymbol{j}}$.



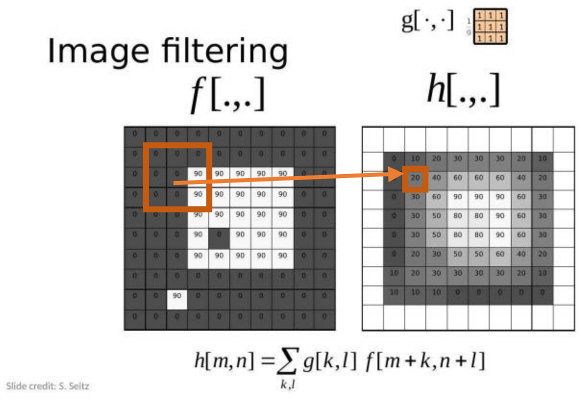
- ***Rotation by 90°*** (clockwise): After applying this operation on an image $\boldsymbol{I}$, the resulting image $\boldsymbol{J}$ in $\mathbb{R}^{h\times w\times 3}$ (note that the width and height of $\boldsymbol{J}$ are swapped!) is such that $\boldsymbol{J}_{\boldsymbol{i},\boldsymbol{j}} = \boldsymbol{I}_{\boldsymbol{h}-\boldsymbol{j},\boldsymbol{i}}$.



- ***Intensity inversion***: After applying this operation on an image $\boldsymbol{I}$, the resulting image $\boldsymbol{J}$ in $\mathbb{R}^{\boldsymbol{h}\times\boldsymbol{w}\times\boldsymbol{3}}$ is such that $\boldsymbol{J}_{\boldsymbol{i},\boldsymbol{j}} = \boldsymbol{M} - \boldsymbol{I}_{\boldsymbol{i},\boldsymbol{j}}$. where the subtraction is componentwise.

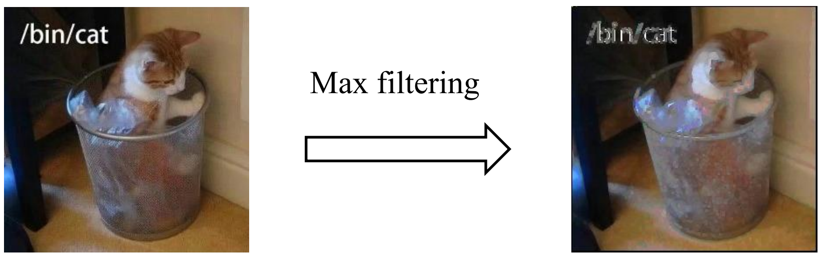


- ***Filtering***: This operation consists in computing the new value of each pixel with an aggregating function applied on a local region centered around that pixel. A simple case is when this aggregating function is the mean (simple average). The resulting image would be a smoothed version of the input image. Other aggregating functions can be considered, such as **max** or **median**, with different effects on the output image. Formally, the resulting image $\boldsymbol{J} \in \mathbb{R}^{\boldsymbol{w}\times\boldsymbol{h}\times\mathbf{3}}$ is given by: $\boldsymbol{J_{i,j}} = \boldsymbol{agg}\{\boldsymbol{I_{i+k,j+l}} \mid -\boldsymbol{s} \leq \boldsymbol{k} \leq \boldsymbol{s}, -\boldsymbol{s} \leq \boldsymbol{l} \leq \boldsymbol{s}\}$ where $\boldsymbol{agg}$ is an aggregating function that is applied on a set of values in a local region defined by a $(\mathbf{2}\boldsymbol{s}+\mathbf{1}) \times (\mathbf{2}\boldsymbol{s}+\mathbf{1})$ square centered around the pixel at $(\boldsymbol{i},\boldsymbol{j})$. Note $\boldsymbol{agg}$ is applied componentwisely over the RGB values. If some indices become negative, the corresponding RGB vector is assumed to be the zero vector. **Important**: You can assume that $\boldsymbol{s} = \mathbf{1}$ in this project.

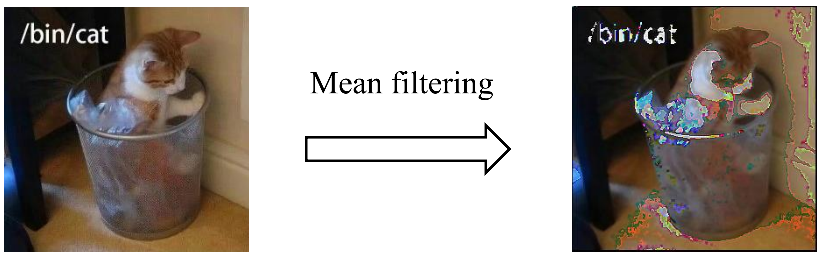


We consider three possible cases for $\boldsymbol{agg}$, which leads to three types of filtering:

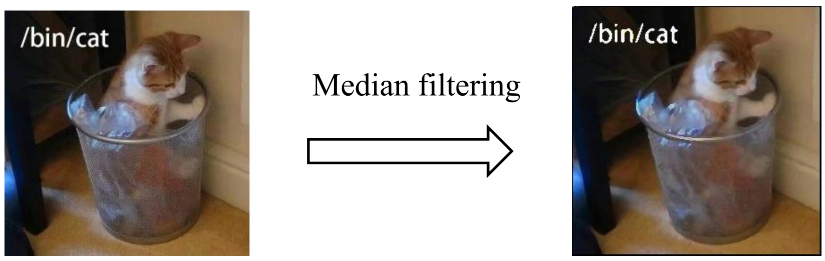
  - *Max filtering*: The resulting image $\boldsymbol{J} \in \mathbb{R}^{\boldsymbol{w}\times\boldsymbol{h}\times\mathbf{3}}$ is given by: $\boldsymbol{J_{i,j}} = \boldsymbol{max}\{\boldsymbol{I_{i+k,j+l}} \mid -\boldsymbol{s} \leq \boldsymbol{k} \leq \boldsymbol{s}, -\boldsymbol{s} \leq \boldsymbol{l} \leq \boldsymbol{s}\}$. Note the $\boldsymbol{max}$ operation is componentwise over the RGB values.



  - *Mean filtering*: The resulting image $\boldsymbol{J} \in \mathbb{R}^{\boldsymbol{w}\times\boldsymbol{h}\times\mathbf{3}}$ is given by: $\boldsymbol{J_{i,j}} = \boldsymbol{mean}\{\boldsymbol{I_{i+k,j+l}} \mid -\boldsymbol{s} \leq \boldsymbol{k} \leq \boldsymbol{s}, -\boldsymbol{s} \leq \boldsymbol{l} \leq \boldsymbol{s}\}$ where the $\boldsymbol{mean}$ operation is also componentwise over the RGB values.



- *Median filtering*: The resulting image $\boldsymbol{J} \in \mathbb{R}^{w\times h\times 3}$ is given by: $\boldsymbol{J}_{i,j,c} = \boldsymbol{median}\{\boldsymbol{I}_{i+k,j+l,c} \mid -\boldsymbol{s} \leq \boldsymbol{k} \leq \boldsymbol{s}, -\boldsymbol{s} \leq \boldsymbol{l} \leq \boldsymbol{s}\}$, for $0 \leq c \leq 3$ representing the three channels (R, G, B) of the image. Recall that the median of a list of values is the value that separates that list in half, one half being smaller and one half being larger than the median.



Part II. File Formats

Many image file formats exist, such as `jpg`, `png`, or `gif`. These file formats describe how an image is stored in a file, usually after applying some compression algorithm to reduce the overall file size. In this project, we use a format called portable pixmap format (PPM), whose file extension is `ppm`. This image file format stores images without any compression. Normally, most image viewers (e.g., ImageMagick, MacOs Preview) can deal with this type of image files.

A PPM file is composed of two parts: a header and its image content. For an RGB image $\boldsymbol{I}$ as discussed above, the header is formatted as follows:

```
P6
w h
M
```

The header is directly followed by the image content:

```
RGBRGB……
RGBRGB……
```

where `P6` is a code to specify that the image content is stored in binary, w, h, and M are as defined above. As a remark, if `P3` is used as a code, the image content is stored in ASCII. After the header part, the file contains the list of the RGB values written in binary. Among the starter files, you can find in `ppm.cpp` a simple example that writes a PPM file. You can use that C++ code to help you understand better the PPM format.

Important: The PPM files that you will generate should be in the same format as in this C++ code. Moreover, your program will only deal with PPM files written with the code `P6`. The binary encoding can reduce the file sizes of the images you will deal with. Moreover, you can assume that $w \leq 800$, $h \leq 800$, and $M = 255$. Those three constants are denoted `WMAX`, `HMAX`, and `M` respectively in the code snippets below.

IV. Programming Assignment

We describe in this section the different C++ functions you need to implement. Your `main` function will read potential options passed to it and process the input image, if any. We describe the behavior of the `main` function at the end. To help you understand better, we ask you some questions (see Self-Quiz below) that you can answer for yourself. You do not need to submit your answers to us.

We assume that a pixel takes values of the following type:

```
typedef struct{
  char red;
  char green;
  char blue;
} Rgb;
```

Since we have not learned dynamic memory management yet, you will create images of the maximum sizes. Therefore, an image will be of the following type:

```
typedef struct{
  Rgb image[WMAX][HMAX];
  unsigned int w;
  unsigned int h;
} Image;
```

We call a value of this type an image array.

First, you need to take care of the input and output of your program:

- implement a function that reads an image from an input stream (e.g., file or standard input). This function takes two arguments: an input stream and an image array. The image is read from the input stream and is stored in the second argument. The caller can then access this image from the image array. The signature of this function is:

  ```
  void readImage(std::istream &is, Image &imI);
  ```

Self-Quiz: Why is the image passed by reference? Could we remove `&` since an array is inside? How else could we have passed `imI`?

- implement a function that writes an image array to an output stream (e.g., file or standard output). This function takes two arguments: an output stream and an image array. The image array is written in the output stream using the PPM file format described above. The signature of this function is:

```
void writeImage(std::ostream &os, const Image &imI);
```

Self-Quiz: Why is `imI` passed like this?

Next, you need to implement the different image transformations. Each of them is performed by its corresponding C++ function. All these functions take at least two arguments: an input image and an output image. The caller can obtain the resulting image after the application of an image transformation from the output image. The signature of an image transformation function `fun` with only two arguments is:

```
void fun(const Image &imI, Image &imJ);
```

Self-Quiz: Could we have returned the resulting image via a return (assuming that we do not know how to do memory allocation)?

- implement a function `verticalFlip` that vertically flips an image.
- implement a function `rotate90` that rotates an image by 90° (clockwise).
- implement a function `intensityInversion` that inverts the intensity of the RGB values.
- implement a function `filter` that applies the filtering operation. Since it depends on an aggregating function, its signature is:

  ```
  void filter(const Image &imI, Image &imJ, Agg f);
  ```

  where the type of the aggregating function is defined by:

  ```
  typedef char (*Agg)(const char values[2s+1][2s+1]);
  ```

  which corresponds to a function pointer that takes a 2D array of chars and return an aggregated value as a char. You will implement three instantiations of aggregating functions: `max`, `mean`, and `median`.

Last, you need to code your `main` function, which will call your previous functions depending on its program arguments. Your program should work according to the following syntax:

```
Usage:  mip [-i input file] [-o output file] -t transformation
```

We will call this previous line the help message.

Recall that in this help message, options in brackets mean that they are optional. The allowed transformations are `verticalFlip, rotate90, intensityInversion, maxFilter, meanFilter,` and `medianFilter`.

Once the program is called with option `--help` or `-h`, even if it is called with other options, the help message should be printed and the program should stop without creating an output image.

If the program is called with incorrect arguments, the pro gram should stop and print the corresponding error message. There is at most one incorrect argument in each command.

Incorrect arguments correspond to the following cases:

- The specified input file does not exist. Error message:

  ```
  Error: The specified input file does not exist.
  ```

- The specified input file exists, but is not a PPM file. Error message:

  ```
  Error: The specified input file exists, but is not a PPM file.
  ```

- The specified transformation does not correspond to any accepted transformations. Error message:

  ```
  Error: The specified transformation does not correspond to any
  accepted transformations.
  ```

Important: When you print any messages, end it with a newline. In any case, do not print any other messages, or your grading on JOJ will be penalized.

Examples:

- Command:

  ```
  ./mip -t invalid_transform -i test.ppm -o test_out.ppm
  ```

  Output:

  ```
  Error: The specified transformation does not correspond to any
  accepted transformations.
  ```

- Command:

  ```
  ./mip -t verticalFlip -i test.txt -o test_out.ppm
  ```

  Output:

  ```
  Error: The specified input file exists, but is not a PPM file.
  ```

- Command:

  ```
  ./mip --help -abc -def
  ```

  Output:

  ```
  Usage: mip [-i input file] [-o output file] -t transformation
  ```

VI. Implementation Requirements and Restrictions

1. When writing your code, you may use the following standard header files: `<iostream>`, `<fstream>`, `<sstream>`, `<string>`, `<cstdlib>`, `<vector>` and `<algorithm>`. No other header files can be included.
2. All required output should be sent to the standard output stream; none to the standard error stream.

VII. Source Code Files and Compiling

To compile, you should have `constants.h`, `mip.cpp`, `image.h` in your directory. Use the following Linux command to compile:

```
g++ --std=c++17 -o mip mip.cpp -Wall -Werror
```

We use some features of C++ 17 to implement debug functions. Be sure to use the following option `--std=c++17` when compiling your program.

In order to guarantee that the TAs can compile your program successfully, you should name you source code files exactly like how they are specified above. For this project, as usual, the penalty for code that does not compile will be **severe**, regardless of the reason.

VIII. Submitting and Due Date

You should submit the source code files (in one compressed file) via Online Judge. The due time is 11:59 pm on July 16th, 2022.

IX. Grading

Your program will be graded along three criteria:

1. Functional Correctness
2. Implementation Constraints
3. General Style

Functional Correctness is determined by running a variety of test cases against your program, checking against our reference solution. We will grade Implementation Constraints to see if you have met all of the implementation requirements and restrictions. General Style refers to the ease with which TAs can read and understand your program, and the cleanliness and elegance of your code. For example, significant code duplication will lead to General Style deductions.