Dennis Ritchie's Secret!

§1 The Goal

The following might look like a regular photo of Dennis Ritchie, who is best known as the creator of the C programming language. But it is not! The photo contains a hidden message. Figuring out this message is the goal of this assignment.

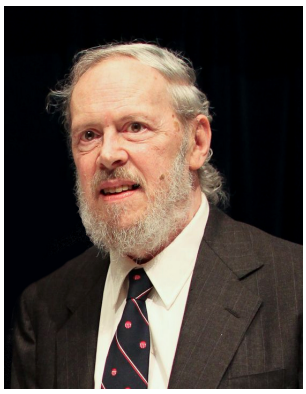

Figure 1: Dennis Ritchie, the creator of C, has a secret.

§2 The PPM Format

In this assignment, we will be working with `.ppm` files. `.ppm` is a simple format for storing images. You are probably more familiar with formats like `.jpg` and `.png`. But as we will soon see, `.ppm` has some advantages over them.

Opening a PPM File

In order to work with PPM files, you will need a PPM viewer.

1. If you are on a Linux machine, you should already have a PPM viewer installed. Go to 3.
2. If you are on a Windows machine, we recommend IrfanView. Download IrfanView from here: https://www.irfanview.net/download_sites.htm. Select SourceForge.
3. Open `DennisRitchie.ppm` and `toy.ppm` using your PPM viewer of choice.

If you have done everything correctly up to this point, you should be able to view PPM images. If you open `DennisRitchie.ppm`, you should be able to see an image of Dennis Ritchie. This

image has a hidden secret message that you will be trying to decipher. If you open `toy.ppm`, you should be able to see an image similar to the following. `toy.ppm` is very small (only 3 pixels wide and 2 pixels tall). So, you might have to zoom in a lot before you can actually see anything. And even when you see something, it won't be exactly similar. It will be a blurrier version of the image below. But you should still be able to see which color goes where.

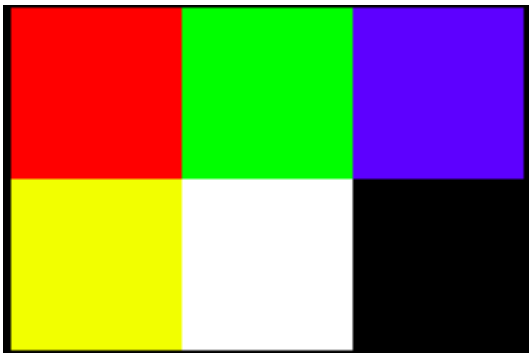

Figure 2: `toy.ppm` magnified

The advantage of PPM files over `.jpg` or `.png` files is that you can open them using a text editor and be able to tell a lot of things. Open `toy.ppm` using your favorite text editor (like `Notepad`), and you should be able to see something like this:

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

- The first line, `P3`, tells us that this is a file in PPM format.
- The second line tells us the width and the height of our image. In this case, our image is 3 pixels wide and 2 pixels tall.
- The third line, `255`, tells us the maximum color value. In our case, a value of 255 implies that color values will range from 0 to 255.
- The rest of the file describes the actual pixels in the image - each pixel is described by three numbers, representing the red, green, and blue values in that pixel. The content in the body of the file confirms the information in the header: since each pixel needs 3 numbers, we have 3 pixels per row and 2 rows.
- The first pixel is described by `255 0 0`. This tells us that the top left pixel should be red. Make sure you understand why the other pixels are colored the way they are.

Answering Questions about PPM Files

Open `DennisRitchie.ppm` with a text editor like Notepad. You should be able to answer the following questions.

- What is the width and the height of the image?
- What color is the top-left pixel?

§3 The Tasks

§3.1 Task 0 (Reading `utils.h`)

In this assignment, you will need to read in PPM files and convert them to an internal representation that you can use to accomplish the various tasks you are given. In `utils.h`, we have defined some structs that will help you. Your first task is to read these struct definitions.

```
typedef struct Color
{
    int R;
    int G;
    int B;
} Color;


typedef struct Image
{
    Color **image;
    int rows;
    int cols;
} Image;
```

Each pixel in a given PPM file should be stored as a `Color`, and each overall image should be stored as an `Image`. The attributes (`R`, `G`, `B`) for `Color` should respectively be the red, green, and blue values for the stored pixel. For `Image`, the `rows` and `cols` attributes should store the dimensions of the image. The `image` attribute should be a list of `Color*`, where each `Color*` stores the color of a single pixel.

§3.2 Task 1 (Reading and Writing PPM Files)

Your next task is to write the `read_PPM()`, `write_PPM()`, and `free_image()` functions in `main.c`. The comments in `main.c` tell you exactly what you need to do. We duplicate them here for convenience.

```
Image *read_PPM(char *filename)
{
    /* opens a PPM file, constructs an Image object and returns
    a pointer to it. Use fopen(), fscanf(), fprintf(), and fclose().*/
}
```

```
void write_PPM(Image *image, char *filename)
{
    /* takes an Image object and writes to filename in PPM format.*/
}
```

```
void free_image(Image *image)
{
    /* takes an Image object and frees all the memory associated with it.
    This involves not only calling free on image but also on the appropriate
    members of it.
    */
}
```

For debugging purposes, call `read_PPM()` on `toy.ppm`, call `write_PPM()` on the output you get from it, and check if the new file you get is identical to the original file.

§3.3 Task 2 (Deciphering the Hidden Message)

This is the part where we tell you how the secret message is hidden. The secret message is itself a black and white PPM image of the same dimensions as `DennisRitchie.ppm`. This means each pixel of the secret message is either black or white. If a pixel is black in the secret image, then the least significant bit of the `B` channel of the corresponding pixel in `DennisRitchie.ppm` is equal to 0. Otherwise, the bit is equal to 1. So, in order to decipher the message, all you need to do is look at the least significant bit of the `B` channel of each pixel in `DennisRitchie.ppm` and decide if the corresponding pixel in the secret image is black or white.

For this task, you will write the functions `evaluate_one_pixel()`, `get_secret_image()`, and `main()` to decipher the hidden message. As always, the comments tell you exactly what to do.

```
Color *evaluate_one_pixel(Image *image, int row, int col)
{
    /* takes an Image object and returns what color the pixel
    at the given row/col should be in the secret image. This
    function should not change image*/
}
```

```
Image *get_secret_image(Image *image)
{
    /* takes an Image object, and constructs the secret image from it by
    extracting the LSB of the B channel.
    You should call evaluate_one_pixel() here. */
}
```

```
int main()
{
    /* Call read_PPM(), write_PPM(), free_image(), get_secret_image()
    in some order to obtain the hidden message.*/

    return 0;
}
```

§4 Submission Instructions

Write everything you need to in `main.c`, put `main.c`, and the secret message inside a folder, name that folder as your student ID, zip it, and submit it to Moodle.

The deadline for submission is April 1, 11:59 pm.

§5 Acknowledgements

A significant portion of this assignment is inspired by Dan Garcia from University of California Berkeley.

§6 Notes

- Make sure you follow the submission instructions. Failure to follow them will result in penalties.
- If you are able to decipher the secret message, please do not spoil it for others. A big part of the enjoyment of solving puzzles like these comes from doing it by yourself.