EE231002 Introduction to Programming

Lab14. Image Processing

Due: Jan. 9, 2021

Today's digital images are composed of individual picture elements, called pixels. With a large number of pixels, the image can be clear. But, with a large number of pixels the image file would need a larger disk space for storage. Thus, most of the popular image file standards involve some compression scheme to reduce the storage overhead. In this lab, we'll concentrate on one of the simplest color image format, portable pixmap format (PPM).

An image can be represented by a two-dimension array of pixels, the x-direction is the width and y-direction is the height of the image. In PPM, each pixel is represented by 3 bytes for red, green and blue colors. Thus, the total number of color levels is 2^{24}, ∼16M. The format of a PPM file is

```
P6
W H
255
```
$\mathtt{R_0G_0B_0R_1G_1B_1} \cdots \mathtt{R_NG_NB_N}$

Where `W` and `H` are two integers and 255 is the intensity level for each color components (R, G, and B). Since each color component R, G, B takes one byte, the intensity level is 255. After the header, the remaining file consists of `3N`, `N`=`W`×`H`, bytes, each byte representing the intensity of one of the color components. This file can be read as following: the first line read using a string of two characters, the second line read using `"%d %d"` for two integers, the third line read using a string of 3 characters, then the remaining file read using `3N` characters. Note that the new-line character after the 3rd line should be handled carefully to ensure the image are read in correctly. Also, the pixels are arranged in column-major fashion, instead of row-major matrix storage use by `C` compiler.

In this lab, you will need to read in an image file and the EE Department logo, then change the shirt color and add the EE logo to the top center position. Examples of program output are shown at the end of this file.

Note that when $R = G = B$ at a pixel, this pixel has a gray color. And the levels of gray scale is 256. When $R = G = B$ =255, the color is white and when $R = G = B = 0$, the pixel has the color black. When placing EE Department logo, the pixels of the original image are simply replaced by the logo pixels *when the logo pixel is not white.*

The data structure for the images should be as following:

```
typedef struct sPIXEL {      // a single pixel
    unsigned char r, g, b;   // three color components
} PIXEL;

typedef struct sIMG {        // an image of PPM style
    char header[3];          // header, either P3 or P6
    int W, H;                // width and height of the image
    int level;               // intensity level of each color component
    PIXEL **PX;              // two-dimensional array for all the pixels
} IMG;
```

Your program should have the following functions:

1. `IMG *PPMin(char *inFile);`
   This function opens the `inFile`, reads the image data and returns a pointer pointing to the newly created image data structure.

2. `void PPMout(IMG *p1, char *outFile);`
   This function writes the image pointed by `p1` to the output file `outFile`.

3. `IMG *PPMcvt(IMG *p1, IMG *ee);`
   This function processes the image pointed by `p1` performing the modifications stated above and returns the new image as a result.

The image file is given as `pic1.ppm`. The EE Department logo is also given in `EE.ppm`. Since these `ppm` files take a large disk space, do not copy them to your own directory.

Your program needs to read an image file and EE Department logo to produce an output file. Thus, the execution of your program should be invoked by the following command line

```
$ ./a.out ~ee2310/lab14/pic1.ppm ~ee2310/lab14/EE.ppm new.ppm
```

Where `new.ppm` is the resulting output file.

In order to read from a file, a `FILE` variable needs to be declared, the file opened and assigned to the file variable, then `fscanf` can be used to read and `fprintf` for write. After all data have been read and written, those file variables should be `close`d. The following example shows reading a integer from a file `data.in` and write to `data.out`.

```
FILE *fin, *fout;
int k;

fin = fopen("data.in", "r");
fout = fopen("data.out", "w");
fscanf(fin, "%d", &k);
fprintf(fout,"%d\n", k);
fclose(fout);
fclose(fin);
```

Of course, your can use `fopen(argv[1], "r");` to open a file specified by the command line argument.

Notes.

1. Create a directory **`lab14`** and use it as the working directory.
2. Name your program source file as **`lab14.c`**.
3. The first few lines of your program should be comments as the following.

   ```
   // EE231002 Lab14. Image Processing
   // ID, Name
   // Date:
   ```

4. After you finish verifying your program, you can submit your source code by

   `$ ~ee2310/bin/submit lab14 lab14.c`

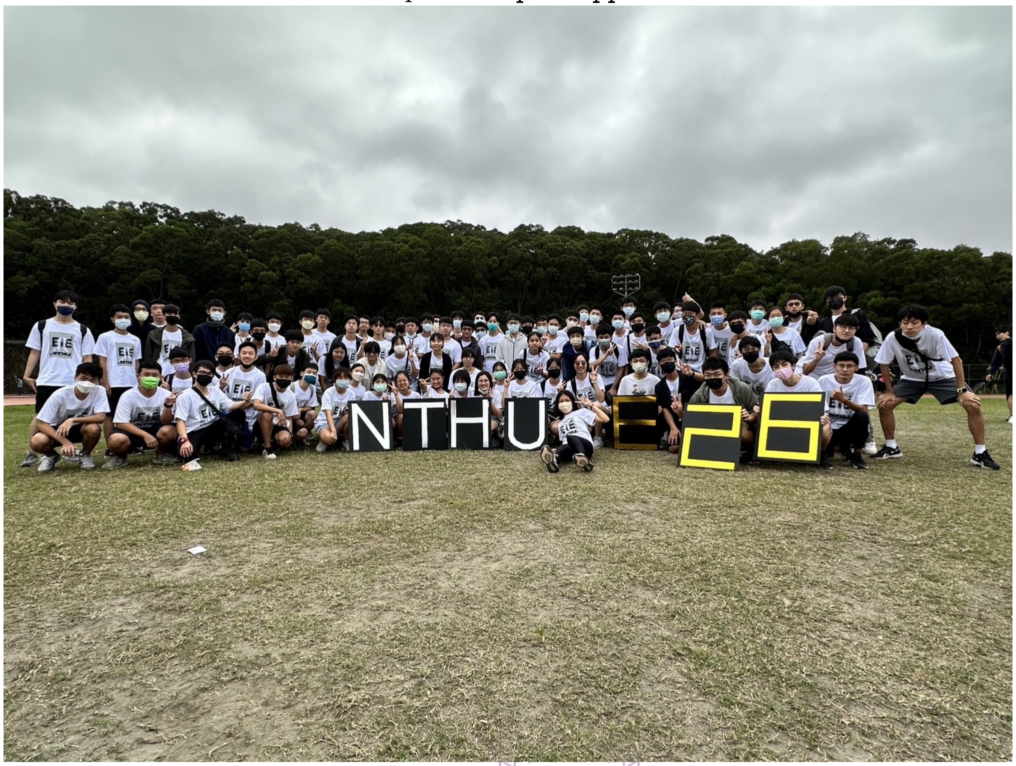
   If you see a ”submitted” message, then you are done. In case you want to check which file and at what time you submitted your labs, you can type in the following command:

   `$ ~ee2310/bin/subrec lab14`

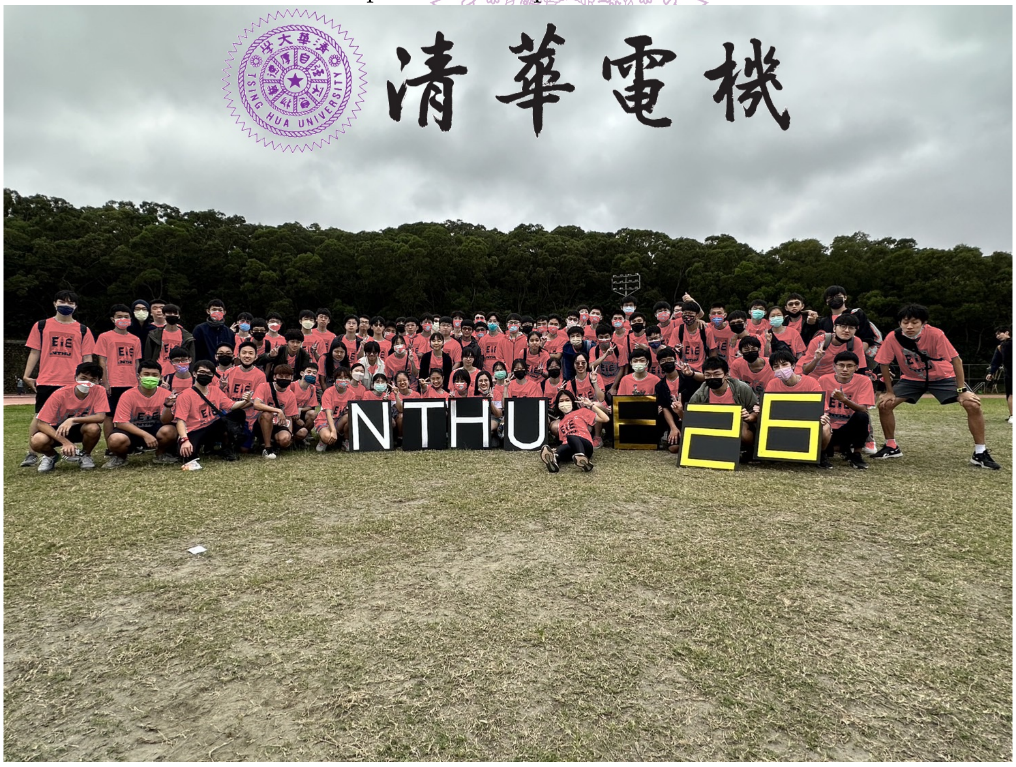
   It will show submission records for lab14.

5. You should try to write the program as efficient as possible. The format of your program should be compact and easy to understand. These are part of the grading criteria.

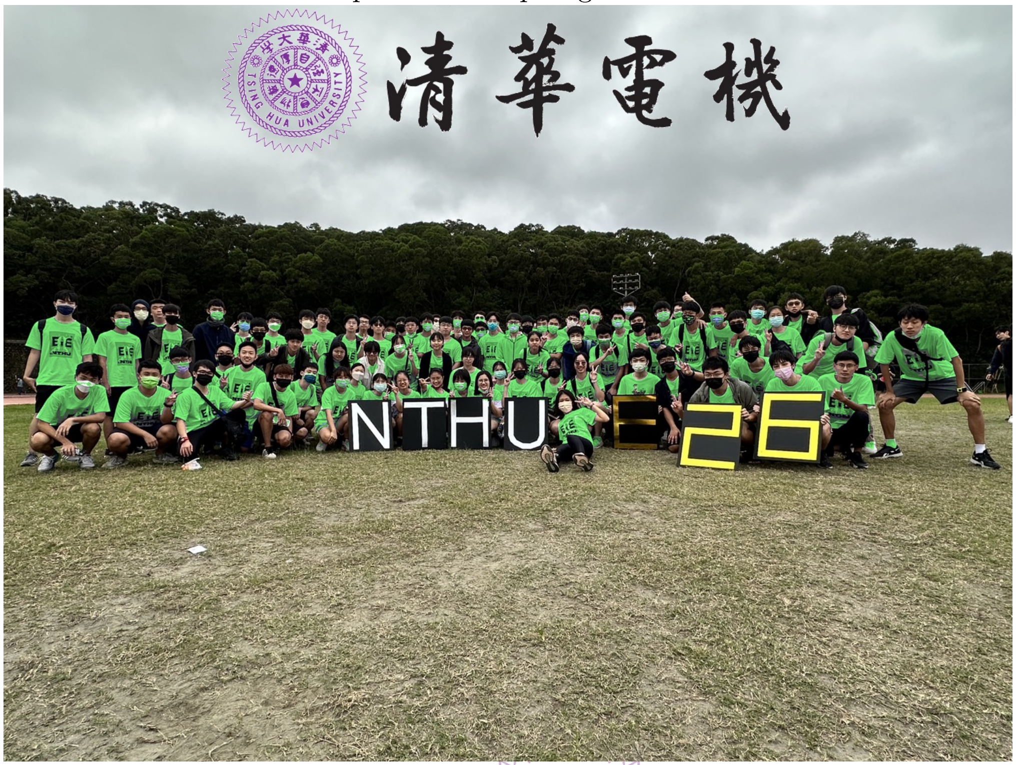
Input file: `pic1.ppm`



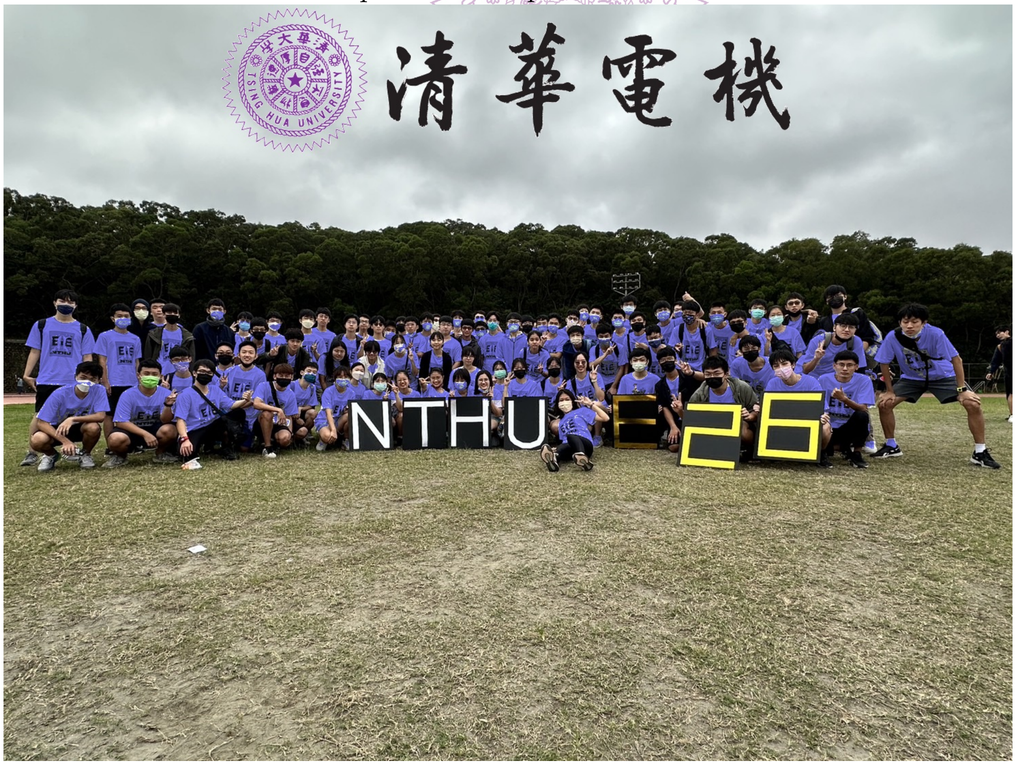
Output file example: red shirt



Output file example: green shirt



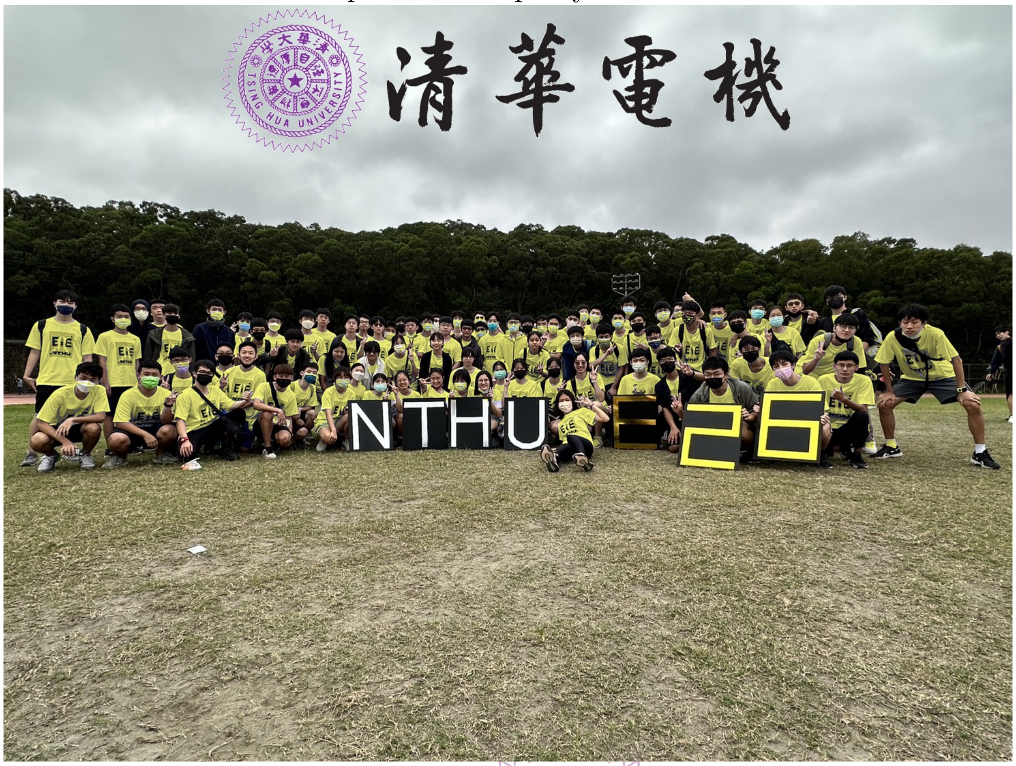
Output file example: blue shirt



Output file example: purple shirt

Output file example: cyan shirt

Output file example: yellow shirt



Output file example: gray shirt