lab14

```
$ gcc lab14.c
$ ./a.out ../pic1.ppm ../EE.ppm ../NTHU.ppm out.ppm
score: 80
o. [Output] Program output is correct, good.
o. [Format] Program format can be improved
o. [Coding] lab14.c spelling errors: dimentional(1), thr(1)
o. [PPMcvt] needs to return a new Image.
o. [Color] box should be defined by x1,y1 and x2,y2 coordinates.
o. [Box] border should be enclosing the color image.
```

lab14.c

```c
// EE2310 Lab14. Image Processing
// 109061217, 林峻霆
// 2020/01/04

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

typedef struct sPIXEL {      // a single pixel
    unsigned char r, g, b;   // three color component
} 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-dimentional array for all the pixels
} IMG;

IMG *PPMin(char *inFile);             // a function read in data from image
                                      // a function process the image data
IMG *PPMcvt(IMG *p1, IMG *ee, IMG *nthu, int x1, int y1, int x2, int y2);
void PPMout(IMG *pic, char *outFile); // a function output the data

int main(int argc, char *argv[])
{
    IMG *p1;                                    // the image of students
    IMG *ee;                                    // NTHUEE logo
    IMG *nthu;                                  // NTHU logo
    int x1, y1, x2, y2;                         // boundary of block

    p1 = PPMin(argv[1]);                        // read in image p1
    ee = PPMin(argv[2]);                        // read in image ee
    nthu = PPMin(argv[3]);                      // read in image nthu
    x1 = 120;                                   // set the boundary
    x2 = p1->w - x1;
    y1 = p1->h - ee->h -150;
    y1 = p1->h - ee->h - 150;
    y2 = nthu->h;

    p1 = PPMcvt(p1, ee, nthu, x1, y1, x2, y2);  // processing thr images
```

```
40
41      PPMout(p1, argv[4]);                            // output the image
42      return 0;                                       // end the program
43  }
44
45  IMG *PPMin(char *inFile)
46      // This function opens an input file, reads in data and stores to a new
47      // data structure
48  {
49      int i, j;    // parameter for loop
50      IMG *pic;    // a data structure storing data of image
51      FILE *fin;   // input file
52
53      pic = (IMG*)malloc(sizeof(IMG));                    // allocate memory
54      fin = fopen(inFile, "r");                           // open file
55      fscanf(fin, "%s", pic->header);                     // read in data
56      fscanf(fin, "%d %d\n%d\n", &pic->w, &pic->h, &pic->level);
57
58      pic->PX = (PIXEL**)malloc(pic->w * sizeof(PIXEL*)); // allocate memory
59      for (i = 0; i < pic->w; i++)
60          pic->PX[i] = (PIXEL*)malloc(pic->h * sizeof(PIXEL*));
61      for (j = 0; j < pic->h; j++) {                      // read in data
62          for (i = 0; i < pic->w; i++) {
63              pic->PX[i][j].r = getc(fin);
64              pic->PX[i][j].g = getc(fin);
65              pic->PX[i][j].b = getc(fin);
66          }
67      }
68
69      fclose(fin);                                        // close the file
70      return pic;                                         // return result
71  }
72
73  IMG *PPMcvt(IMG *p1, IMG *ee, IMG *nthu, int x1,int y1, int x2, int y2)
    IMG *PPMcvt(IMG *p1, IMG *ee, IMG *nthu, int x1, int y1, int x2, int y2)
74      // This function process the data in the following order:
75      //      steps 1. convert image to gray-scale, instead of myself
76      //      steps 2. draw the boundary of the block
77      //      steps 3. put on NTHU logo
78      //      steps 4. put on NTHUEE logo
79      //      steps 5. return processed image
```

```
80 {
81     int i, j;           // parameter for loop and index
82     int x_dir, y_dir;   // the coordinate of NTHUEE logo
83     char gray_level;    // variable for gray-scale converting
84 
85     PIXEL blue;         // a pixel of blue color
86     blue.r = 0;
87     blue.g = 183;
88     blue.b = 255;
89 
90     // convert everyone in the image instead of me into gray-scale.
91     for (i = 0; i < p1->w; i++) {
92         for (j = 0; j < p1->h; j++) {
93             if (i < 1100 || i > 1320 || j < 1350 || j > 1650) {
94                 gray_level = 0.2126 * p1->PX[i][j].r + 0.7152 *
95                             p1->PX[i][j].g + 0.0722 * p1->PX[i][j].b;
96                 p1->PX[i][j].r = gray_level;   // convert to gray-scale
97                 p1->PX[i][j].g = gray_level;   // convert to gray-scale
98                 p1->PX[i][j].b = gray_level;   // convert to gray-scale
99             }
100        }
101    }
102
103    // draw the boundary of block
104    for (j = 0; j < 10; j++) {
105        for (i = x1; i <= x2; i++) {
106            p1->PX[i][y1 + j] = blue;      // set the color blue
107            p1->PX[i][y2 + j] = blue;      // set the color blue
108        }
109    }
110    for (i = 0; i < 10; i++) {
111        for (j = y2; j <= y1; j++) {
112            p1->PX[x1 + i][j] = blue;      // set the color blue
113            p1->PX[x2 + i][j] = blue;      // set the color blue
114        }
115    }
116
117    // put the NTHU logo on the upper-left corner
118    for (i = 0; i < nthu->w; i++) {
119        for (j = 0; j < nthu->h; j++) {
120            if (nthu->PX[i][j].r != 255 && nthu->PX[i][j].g != 255 &&
```

```
121                 nthu->PX[i][j].b != 255) { // check whether pixel is white
122                 p1->PX[i][j].r = 255;      // set the color intensity
123                 p1->PX[i][j].b = 255;
124             }
125         }
126     }
127
128     // put th NTHUEE logo on the bottom-middle part of image
129     x_dir = (p1->w - ee->w) / 2;            // calculate the coordinate
130     y_dir = p1->h - ee->h;                  // calculate the coordinate
131     for (i = 0; i < ee->w; i++) {
132         for (j = 0; j < ee->h; j++) {
133             if (ee->PX[i][j].r != 255 && ee->PX[i][j].g != 255 &&
134                 ee->PX[i][j].b != 255) {    // check if the pixel is white
135                 p1->PX[x_dir + i][y_dir + j - 180] = ee->PX[i][j];
136             }
137         }
138     }
139
140     return p1;                              // return the processed image
141 }
142
143 void PPMout(IMG *pic, char *outFile)
144     // a function output the data from the data structure to output file
145 {
146     int i, j;                                       // parameter for loop
147     FILE *fout;                                     // output file
148
149     fout = fopen(outFile, "w");                     // open the file
150     fprintf(fout, "%s\n%d %d\n%d\n", pic->header, pic->w, pic->h,
151             pic->level);
152     for (j = 0; j < pic->h; j++) {                  // output the data
153         for (i = 0; i < pic->w; i++) {
154             fprintf(fout, "%c", pic->PX[i][j].r);
155             fprintf(fout, "%c", pic->PX[i][j].g);
156             fprintf(fout, "%c", pic->PX[i][j].b);
157         }
158     }
159
160     fclose(fout);                                   // close the file
161 }
```

```
162
```