Fundamental Exercise on Computer and Information Engineering 1B
Image Processing

XL15613 Thiago Machado da Silva

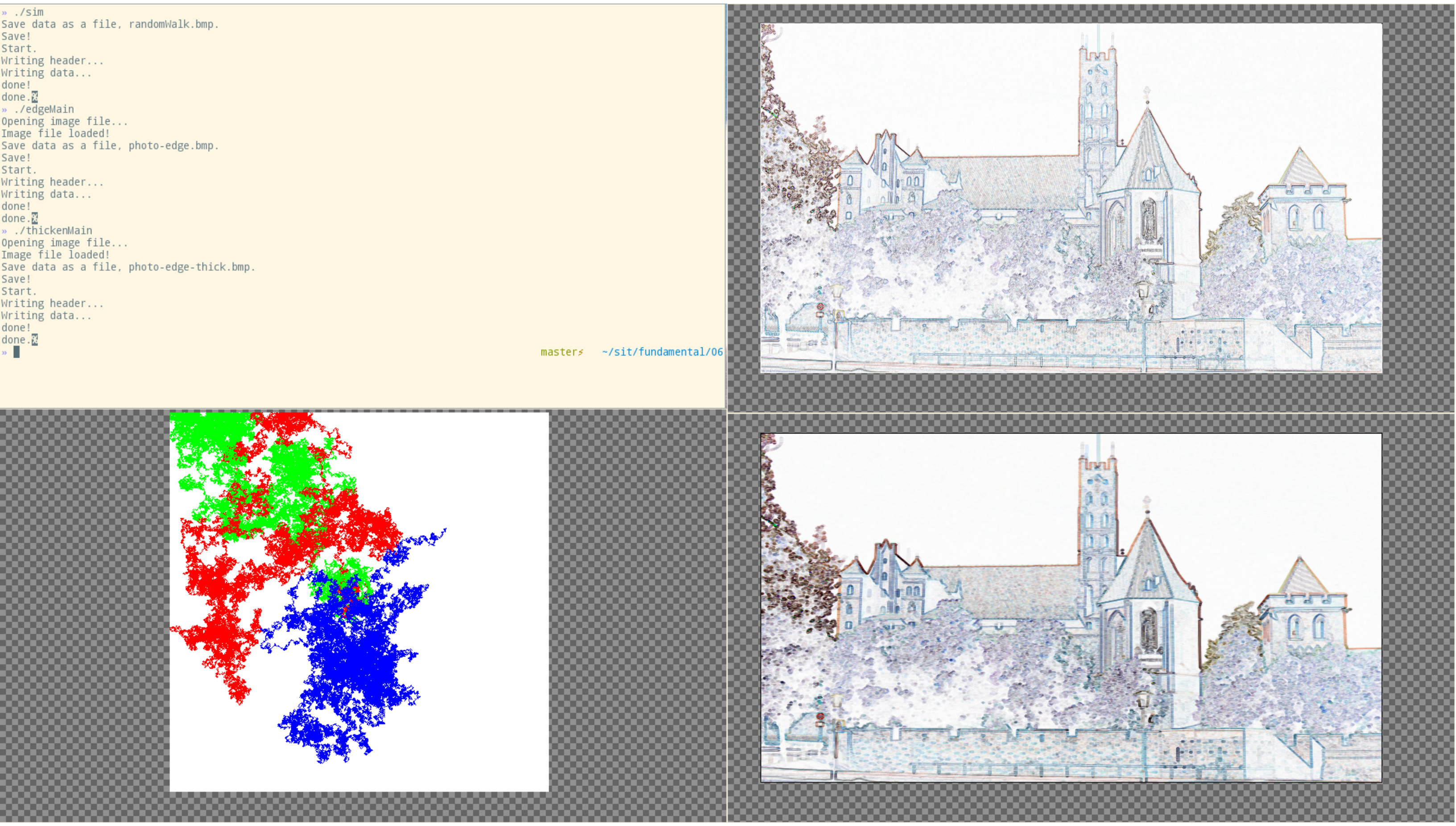
July 21, 2015

Output

In Figure 1 the executions reports no error. The randomWalk, photo-edge and photo-edge-thick images were constructed properly. The randomWalk image reminds me of `xscreensaver's "Wander"` animation, where there is only one walker, but it changes the color from time to time, and it appear in the opposite side of the screen when one side is reached.

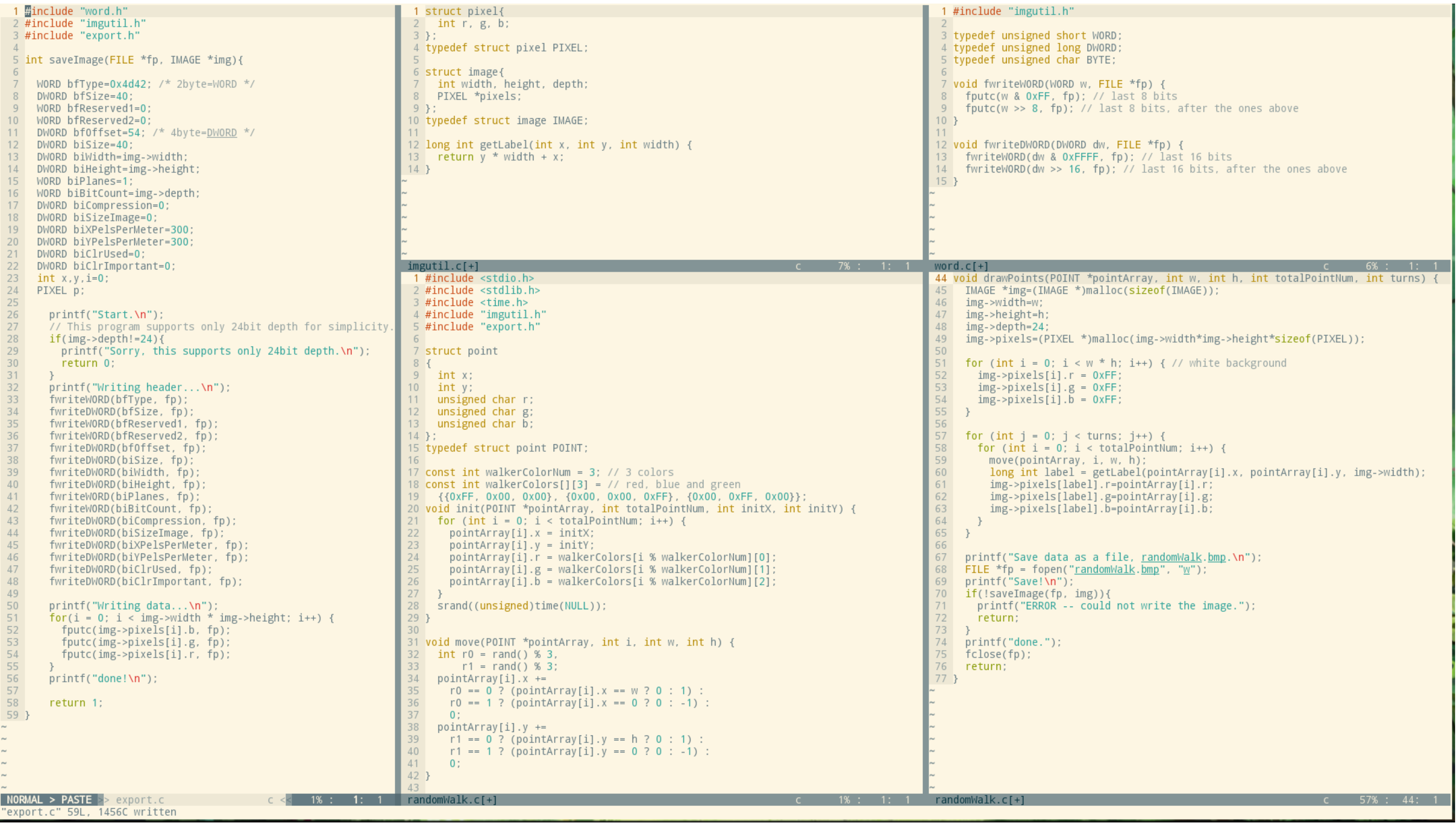
Source codes

Shown in Figure 2 and Figure 3.



```
» ./sim
Save data as a file, randomWalk.bmp.
Save!
Start.
Writing header...
Writing data...
done!
done.%
» ./edgeMain
Opening image file...
Image file loaded!
Save data as a file, photo-edge.bmp.
Save!
Start.
Writing header...
Writing data...
done!
done.%
» ./thickenMain
Opening image file...
Image file loaded!
Save data as a file, photo-edge-thick.bmp.
Save!
Start.
Writing header...
Writing data...
done!
done.%
»                                                                   master⚡   ~/sit/fundamental/06
```

Figure 1: from top to bottom, left to right: programs' outputs, `randomWalk.bmp`, `photo-edge.bmp` and `photo-edge-thick.bmp`.



```c
 1 #include "word.h"
 2 #include "imgutil.h"
 3 #include "export.h"
 4
 5 int saveImage(FILE *fp, IMAGE *img){
 6
 7   WORD bfType=0x4d42; /* 2byte=WORD */
 8   DWORD bfSize=40;
 9   WORD bfReserved1=0;
10   WORD bfReserved2=0;
11   DWORD bfOffset=54; /* 4byte=DWORD */
12   DWORD biSize=40;
13   DWORD biWidth=img->width;
14   DWORD biHeight=img->height;
15   WORD biPlanes=1;
16   WORD biBitCount=img->depth;
17   DWORD biCompression=0;
18   DWORD biSizeImage=0;
19   DWORD biXPelsPerMeter=300;
20   DWORD biYPelsPerMeter=300;
21   DWORD biClrUsed=0;
22   DWORD biClrImportant=0;
23   int x,y,i=0;
24   PIXEL p;
25
26     printf("Start.\n");
27     // This program supports only 24bit depth for simplicity.
28     if(img->depth!=24){
29       printf("Sorry, this supports only 24bit depth.\n");
30       return 0;
31     }
32     printf("Writing header...\n");
33     fwriteWORD(bfType, fp);
34     fwriteDWORD(bfSize, fp);
35     fwriteWORD(bfReserved1, fp);
36     fwriteWORD(bfReserved2, fp);
37     fwriteDWORD(bfOffset, fp);
38     fwriteDWORD(biSize, fp);
39     fwriteDWORD(biWidth, fp);
40     fwriteDWORD(biHeight, fp);
41     fwriteWORD(biPlanes, fp);
42     fwriteWORD(biBitCount, fp);
43     fwriteDWORD(biCompression, fp);
44     fwriteDWORD(biSizeImage, fp);
45     fwriteDWORD(biXPelsPerMeter, fp);
46     fwriteDWORD(biYPelsPerMeter, fp);
47     fwriteDWORD(biClrUsed, fp);
48     fwriteDWORD(biClrImportant, fp);
49
50     printf("Writing data...\n");
51     for(i = 0; i < img->width * img->height; i++) {
52       fputc(img->pixels[i].b, fp);
53       fputc(img->pixels[i].g, fp);
54       fputc(img->pixels[i].r, fp);
55     }
56     printf("done!\n");
57
58     return 1;
59 }
```

```
NORMAL > PASTE >> export.c                c <<  1% :   1:  1
"export.c" 59L, 1456C written
```

```c
 1 struct pixel{
 2   int r, g, b;
 3 };
 4 typedef struct pixel PIXEL;
 5
 6 struct image{
 7   int width, height, depth;
 8   PIXEL *pixels;
 9 };
10 typedef struct image IMAGE;
11
12 long int getLabel(int x, int y, int width) {
13   return y * width + x;
14 }
```

```
imgutil.c[+]                                  c      7% :   1:  1
```

```c
 1 #include "imgutil.h"
 2
 3 typedef unsigned short WORD;
 4 typedef unsigned long DWORD;
 5 typedef unsigned char BYTE;
 6
 7 void fwriteWORD(WORD w, FILE *fp) {
 8   fputc(w & 0xFF, fp); // last 8 bits
 9   fputc(w >> 8, fp); // last 8 bits, after the ones above
10 }
11
12 void fwriteDWORD(DWORD dw, FILE *fp) {
13   fwriteWORD(dw & 0xFFFF, fp); // last 16 bits
14   fwriteWORD(dw >> 16, fp); // last 16 bits, after the ones above
15 }
```

```
word.c[+]                                     c      6% :   1:  1
```

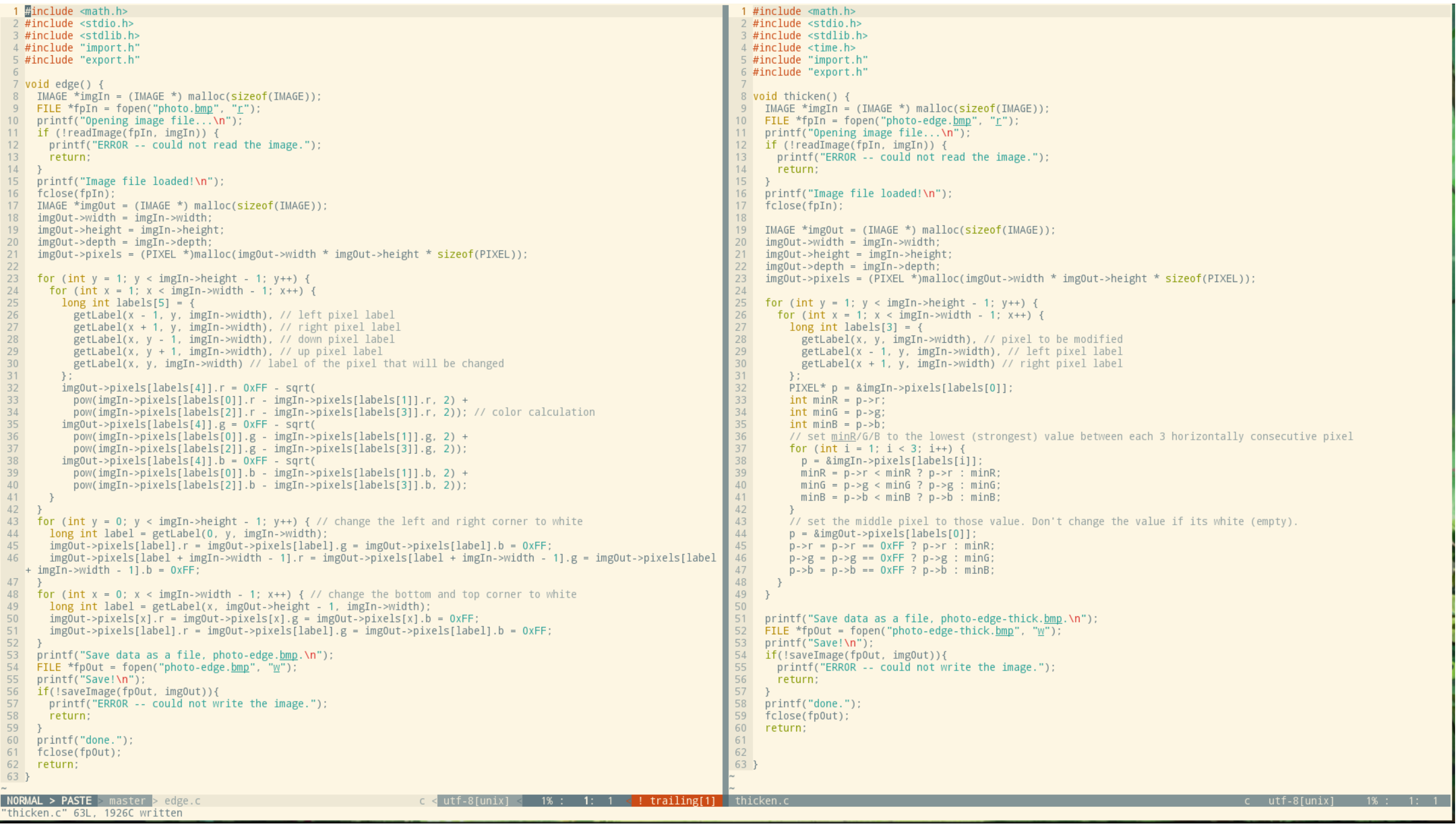
```c
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #include <time.h>
 4 #include "imgutil.h"
 5 #include "export.h"
 6
 7 struct point
 8 {
 9   int x;
10   int y;
11   unsigned char r;
12   unsigned char g;
13   unsigned char b;
14 };
15 typedef struct point POINT;
16
17 const int walkerColorNum = 3; // 3 colors
18 const int walkerColors[][3] = // red, blue and green
19   {{0xFF, 0x00, 0x00}, {0x00, 0x00, 0xFF}, {0x00, 0xFF, 0x00}};
20 void init(POINT *pointArray, int totalPointNum, int initX, int initY) {
21   for (int i = 0; i < totalPointNum; i++) {
22     pointArray[i].x = initX;
23     pointArray[i].y = initY;
24     pointArray[i].r = walkerColors[i % walkerColorNum][0];
25     pointArray[i].g = walkerColors[i % walkerColorNum][1];
26     pointArray[i].b = walkerColors[i % walkerColorNum][2];
27   }
28   srand((unsigned)time(NULL));
29 }
30
31 void move(POINT *pointArray, int i, int w, int h) {
32   int r0 = rand() % 3,
33       r1 = rand() % 3;
34   pointArray[i].x +=
35     r0 == 0 ? (pointArray[i].x == w ? 0 : 1) :
36     r0 == 1 ? (pointArray[i].x == 0 ? 0 : -1) :
37     0;
38   pointArray[i].y +=
39     r1 == 0 ? (pointArray[i].y == h ? 0 : 1) :
40     r1 == 1 ? (pointArray[i].y == 0 ? 0 : -1) :
41     0;
42 }
43
```

```
randomWalk.c[+]                               c      1% :   1:  1
```

```c
44 void drawPoints(POINT *pointArray, int w, int h, int totalPointNum, int turns) {
45   IMAGE *img=(IMAGE *)malloc(sizeof(IMAGE));
46   img->width=w;
47   img->height=h;
48   img->depth=24;
49   img->pixels=(PIXEL *)malloc(img->width*img->height*sizeof(PIXEL));
50
51   for (int i = 0; i < w * h; i++) { // white background
52     img->pixels[i].r = 0xFF;
53     img->pixels[i].g = 0xFF;
54     img->pixels[i].b = 0xFF;
55   }
56
57   for (int j = 0; j < turns; j++) {
58     for (int i = 0; i < totalPointNum; i++) {
59       move(pointArray, i, w, h);
60       long int label = getLabel(pointArray[i].x, pointArray[i].y, img->width);
61       img->pixels[label].r=pointArray[i].r;
62       img->pixels[label].g=pointArray[i].g;
63       img->pixels[label].b=pointArray[i].b;
64     }
65   }
66
67   printf("Save data as a file, randomWalk.bmp.\n");
68   FILE *fp = fopen("randomWalk.bmp", "w");
69   printf("Save!\n");
70   if(!saveImage(fp, img)){
71     printf("ERROR -- could not write the image.");
72     return;
73   }
74   printf("done.");
75   fclose(fp);
76   return;
77 }
```

```
randomWalk.c[+]                               c     57% :  44:  1
```

Figure 2: from left to right, top to bottom: `export.c`, `imageutil.c`, `word.c`, `randomWalk.c` (until line 43) and `randomWalk.c` (from line 44).



```c
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include "import.h"
#include "export.h"

void edge() {
  IMAGE *imgIn = (IMAGE *) malloc(sizeof(IMAGE));
  FILE *fpIn = fopen("photo.bmp", "r");
  printf("Opening image file...\n");
  if (!readImage(fpIn, imgIn)) {
    printf("ERROR -- could not read the image.");
    return;
  }
  printf("Image file loaded!\n");
  fclose(fpIn);
  IMAGE *imgOut = (IMAGE *) malloc(sizeof(IMAGE));
  imgOut->width = imgIn->width;
  imgOut->height = imgIn->height;
  imgOut->depth = imgIn->depth;
  imgOut->pixels = (PIXEL *)malloc(imgOut->width * imgOut->height * sizeof(PIXEL));

  for (int y = 1; y < imgIn->height - 1; y++) {
    for (int x = 1; x < imgIn->width - 1; x++) {
      long int labels[5] = {
        getLabel(x - 1, y, imgIn->width), // left pixel label
        getLabel(x + 1, y, imgIn->width), // right pixel label
        getLabel(x, y - 1, imgIn->width), // down pixel label
        getLabel(x, y + 1, imgIn->width), // up pixel label
        getLabel(x, y, imgIn->width) // label of the pixel that will be changed
      };
      imgOut->pixels[labels[4]].r = 0xFF - sqrt(
        pow(imgIn->pixels[labels[0]].r - imgIn->pixels[labels[1]].r, 2) +
        pow(imgIn->pixels[labels[2]].r - imgIn->pixels[labels[3]].r, 2)); // color calculation
      imgOut->pixels[labels[4]].g = 0xFF - sqrt(
        pow(imgIn->pixels[labels[0]].g - imgIn->pixels[labels[1]].g, 2) +
        pow(imgIn->pixels[labels[2]].g - imgIn->pixels[labels[3]].g, 2));
      imgOut->pixels[labels[4]].b = 0xFF - sqrt(
        pow(imgIn->pixels[labels[0]].b - imgIn->pixels[labels[1]].b, 2) +
        pow(imgIn->pixels[labels[2]].b - imgIn->pixels[labels[3]].b, 2));
    }
  }
  for (int y = 0; y < imgIn->height - 1; y++) { // change the left and right corner to white
    long int label = getLabel(0, y, imgIn->width);
    imgOut->pixels[label].r = imgOut->pixels[label].g = imgOut->pixels[label].b = 0xFF;
    imgOut->pixels[label + imgIn->width - 1].r = imgOut->pixels[label + imgIn->width - 1].g = imgOut->pixels[label + imgIn->width - 1].b = 0xFF;
  }
  for (int x = 0; x < imgIn->width - 1; x++) { // change the bottom and top corner to white
    long int label = getLabel(x, imgOut->height - 1, imgIn->width);
    imgOut->pixels[x].r = imgOut->pixels[x].g = imgOut->pixels[x].b = 0xFF;
    imgOut->pixels[label].r = imgOut->pixels[label].g = imgOut->pixels[label].b = 0xFF;
  }
  printf("Save data as a file, photo-edge.bmp.\n");
  FILE *fpOut = fopen("photo-edge.bmp", "w");
  printf("Save!\n");
  if(!saveImage(fpOut, imgOut)){
    printf("ERROR -- could not write the image.");
    return;
  }
  printf("done.");
  fclose(fpOut);
  return;
}
```

```
"thicken.c" 63L, 1926C written
```

```c
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "import.h"
#include "export.h"

void thicken() {
  IMAGE *imgIn = (IMAGE *) malloc(sizeof(IMAGE));
  FILE *fpIn = fopen("photo-edge.bmp", "r");
  printf("Opening image file...\n");
  if (!readImage(fpIn, imgIn)) {
    printf("ERROR -- could not read the image.");
    return;
  }
  printf("Image file loaded!\n");
  fclose(fpIn);

  IMAGE *imgOut = (IMAGE *) malloc(sizeof(IMAGE));
  imgOut->width = imgIn->width;
  imgOut->height = imgIn->height;
  imgOut->depth = imgIn->depth;
  imgOut->pixels = (PIXEL *)malloc(imgOut->width * imgOut->height * sizeof(PIXEL));

  for (int y = 1; y < imgIn->height - 1; y++) {
    for (int x = 1; x < imgIn->width - 1; x++) {
      long int labels[3] = {
        getLabel(x, y, imgIn->width), // pixel to be modified
        getLabel(x - 1, y, imgIn->width), // left pixel label
        getLabel(x + 1, y, imgIn->width) // right pixel label
      };
      PIXEL* p = &imgIn->pixels[labels[0]];
      int minR = p->r;
      int minG = p->g;
      int minB = p->b;
      // set minR/G/B to the lowest (strongest) value between each 3 horizontally consecutive pixel
      for (int i = 1; i < 3; i++) {
        p = &imgIn->pixels[labels[i]];
        minR = p->r < minR ? p->r : minR;
        minG = p->g < minG ? p->g : minG;
        minB = p->b < minB ? p->b : minB;
      }
      // set the middle pixel to those value. Don't change the value if its white (empty).
      p = &imgOut->pixels[labels[0]];
      p->r = p->r == 0xFF ? p->r : minR;
      p->g = p->g == 0xFF ? p->g : minG;
      p->b = p->b == 0xFF ? p->b : minB;
    }
  }

  printf("Save data as a file, photo-edge-thick.bmp.\n");
  FILE *fpOut = fopen("photo-edge-thick.bmp", "w");
  printf("Save!\n");
  if(!saveImage(fpOut, imgOut)){
    printf("ERROR -- could not write the image.");
    return;
  }
  printf("done.");
  fclose(fpOut);
  return;


}
```

Figure 3: from left to right: `edge.c` and `thicken.c`.