```c
/* ------------------------SECTION 2---------------------*/
#include <math.h>
#include "tiff.h"
#include "allocate.h"
#include "randlib.h"
#include "typeutil.h"
#include <stdio.h>

void error(char *name);

struct pixel {
    int m,n; /* m=row, n=col */
};

void ConnectedNeighbors(
    struct pixel s,
    double T, /* threshold*/
    unsigned char **img, /* 2D array of pixels */
    int width,
    int height,
    int *M, /* a pointer to the number of neighbors connected to the pixel s */
    struct pixel c[4]); /* This is an array containing the M connected neighbors to the pixel s. */

void ConnectedSet(
    struct pixel s,
    double T,
    unsigned char **img,
    int width,
    int height,
    int ClassLabel, /* s the integer value that will be used to label any pixel which is connected to s. */
    uint8_t **seg,
    int *NumConPixels); /* the number of pixels which were found to be connected to s. */

int main (int argc, char **argv)
{
    // Define parameters
    FILE *fp;
    struct TIFF_img input_img, color_img, segmentation;
    // struct pixel s;
    // s.m = 45;
    // s.n = 67;
    double T = 3.0;
    int ClassLabel = 1;
    int NumConPixels = 0;
    int NOR = 0;

    /* accepts a command line argument specifying the value of rho */
    // scanf("%lf", &rho);

    if ( argc != 2 ) error( argv[0] );

    /* open image file */
    if ( ( fp = fopen ( argv[1], "rb" ) ) == NULL ) {
    fprintf ( stderr, "cannot open file %s\n", argv[1] );
    exit ( 1 );
    }

```

```
    /* read image */
    if ( read_TIFF ( fp, &input_img ) ) {
    fprintf ( stderr, "error reading file %s\n", argv[1] );
    exit ( 1 );
    }

    /* close image file */
    fclose ( fp );

    /* check the type of image data */
    if ( input_img.TIFF_type != 'g' ) {
    fprintf ( stderr, "error:  image must be 24-bit color\n" );
    exit ( 1 );
    }

    /* set up structure for output color image */
    /* Note that the type is 'g' rather than 'c' */
    get_TIFF ( &color_img, input_img.height, input_img.width, 'g' );

    // printf("reading success\n");

    /* create a 2D matrix to record segmentations */
    get_TIFF ( &segmentation, input_img.height, input_img.width, 'g' );

    /* Initialize the Output */
    int i;
    int j;
    for ( i = 0; i < input_img.height; i++ ) {
        for ( j = 0; j < input_img.width; j++ ) {
            color_img.mono[i][j] = 0;
        }
    }

    /* Initialize the segmentations */
    for ( i = 0; i < input_img.height; i++ ) {
        for ( j = 0; j < input_img.width; j++ ) {
            segmentation.mono[i][j] = 0;
        }
    }

    // printf("output init success\n");

    i = 0;
    j = 0;
    int ii = 0; int jj = 0;
    for ( i = 0; i < input_img.height; i++ ) {
        for ( j = 0; j < input_img.width; j++ ) {
            struct pixel s;
            s.m = i;
            s.n = j;
            if (color_img.mono[i][j] == 0) {
                ConnectedSet(
                    s,
                    T,
                    input_img.mono,
                    input_img.width,
                    input_img.height,
                    ClassLabel, /* s the integer value that will be used to label any pixel which is connected to s. */
                    segmentation.mono,
```

```
                    &NumConPixels); /* the number of pixels which were found to be connected to s. */
                // printf("%d\n", NumConPixels);
                if (NumConPixels > 100) {
                    NOR = NOR + 1;
                    ClassLabel = ClassLabel + 1;
                    for ( ii = 0; ii < input_img.height; ii++ ) {
                        for ( jj = 0; jj < input_img.width; jj++ ) {
                            color_img.mono[ii][jj] = segmentation.mono[ii][jj] + color_img.mono[ii][jj];
                        }
                    }
                    ii = 0; jj = 0;
                }
            for ( ii = 0; ii < input_img.height; ii++ ) {
                for ( jj = 0; jj < input_img.width; jj++ ) {
                        segmentation.mono[ii][jj] = 0;
                }
            }
            NumConPixels = 0;
            }
        }
    }

    printf("%d\n", NOR);

    // printf("process success\n");

    // i = 0;
    // j = 0;
    // for ( i = 0; i < input_img.height; i++ ) {
    //     for ( j = 0; j < input_img.width; j++ ) {
    //         if (color_img.mono[i][j] == 0) {
    //             color_img.mono[i][j] = 255;
    //         }
    //         else {
    //             color_img.mono[i][j] = 0;
    //         }
    //     }
    // }

    /* open color image file */
    if ( ( fp = fopen ( "color.tif", "wb" ) ) == NULL ) {
        fprintf ( stderr, "cannot open file color.tif\n");
        exit ( 1 );
    }

    /* write color image */
    if ( write_TIFF ( fp, &color_img ) ) {
        fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
        exit ( 1 );
    }

    /* close color image file */
    fclose ( fp );

    /* de-allocate space which was used for the images */
    free_TIFF ( &(input_img) );
    free_TIFF ( &(color_img) );

```

```
175     return(0);
176 }
177
178 void error(char *name)
179 {
180     printf("usage:  %s  image.tiff \n\n",name);
181     printf("this program reads in a 24-bit color TIFF image.\n");
182     printf("It then horizontally filters the green component, adds noise,\n");
183     printf("and writes out the result as an 8-bit image\n");
184     printf("with the name 'green.tiff'.\n");
185     printf("It also generates an 8-bit color image,\n");
186     printf("that swaps red and green components from the input image");
187     exit(1);
188 }
189
190 void ConnectedNeighbors(
191     struct pixel s,
192     double T, /* threshold*/
193     unsigned char **img, /* 2D array of pixels */
194     int width,
195     int height,
196     int *M, /* a pointer to the number of neighbors connected to the pixel s */
197     struct pixel c[4]) /* This is an array containing the M connected neighbors to
    the pixel s. */
198 {
199     if ((s.m > height) || (s.n > width)) {
200         printf("error, the pixel s exceeds the width & height of the image\n");
201     }
202     // *M = 0;
203     if (((s.m-1)>=0) && (abs(img[s.m][s.n] - img[s.m-1][s.n])<=T)) {
204         c[*M].m = s.m - 1;
205         c[*M].n = s.n;
206         *M = *M + 1;
207         // printf("1 \n");
208     }
209     if (((s.m+1)<=height-1) && (abs(img[s.m][s.n] - img[s.m+1][s.n])<=T)) {
210         c[*M].m = s.m + 1;
211         c[*M].n = s.n;
212         *M = *M + 1;
213         // printf("2 \n");
214     }
215     if (((s.n-1)>=0) && (abs(img[s.m][s.n] - img[s.m][s.n-1])<=T)) {
216         c[*M].n = s.n - 1;
217         c[*M].m = s.m;
218         *M = *M + 1;
219         // printf("3 \n");
220     }
221     if (((s.n+1)<=width-1) && (abs(img[s.m][s.n] - img[s.m][s.n+1])<=T)) {
222         c[*M].n = s.n + 1;
223         c[*M].m = s.m;
224         *M = *M + 1;
225         // printf("4 \n");
226     }
227     return;
228 }
229
230 void ConnectedSet(
231     struct pixel s,
232     double T,
233     unsigned char **img,
```

```
234     int width,
235     int height,
236     int ClassLabel, /* s the integer value that will be used to label any pixel
    which is connected to s. */
237     uint8_t **seg,
238     int *NumConPixels) /* the number of pixels which were found to be connected to
    s. */
239 {
240 
241     // printf("processing started\n");
242 
243     int M = 0;
244     *NumConPixels = *NumConPixels + 1;
245     struct pixel c[4];
246     int i=0;
247 
248     // printf("piazza code starts\n");
249     seg[s.m][s.n]=ClassLabel;
250     // Doing recursive is simpler than link-list (idea from Piazza)
251     ConnectedNeighbors(s,T,img,width,height,&M,c);
252     //base case:
253     if(M==0){return;}//no neighbors
254     // if(seg[c[i].m][c[i].n]==1 for all i in range(M)){return;} //all neighbors
    already visited
255     for(i=0;i<M;i++){
256         if(seg[c[i].m][c[i].n]!=ClassLabel) {
257             ConnectedSet(c[i],T,img,width,height,ClassLabel,seg,NumConPixels);
258         }
259         else{
260             continue;
261         }
262     }
263     return;
264 }
```