```c
//-------------------------4 FIR Sharpening Filter-----------------
#include <math.h>
#include "tiff.h"
#include "allocate.h"
#include "randlib.h"
#include "typeutil.h"

void error(char *name);

int main (int argc, char **argv)
{
  FILE *fp;
  struct TIFF_img input_img, green_img, red_img,blue_img, color_img;
  double **img1,**imgr,**imgb,**img2,**img3,**img4,rho;
  int32_t i,j,pixelg,ii,jj,pixelr,pixelb;

  /* 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 != 'c' ) {
    fprintf ( stderr, "error:  image must be 24-bit color\n" );
    exit ( 1 );
  }

  /* Allocate image of double precision floats */
  img1 = (double **)get_img(input_img.width,input_img.height,sizeof(double));
  imgr = (double **)get_img(input_img.width,input_img.height,sizeof(double));
  imgb = (double **)get_img(input_img.width,input_img.height,sizeof(double));
  img2 = (double **)get_img(input_img.width,input_img.height,sizeof(double));
  img3 = (double **)get_img(input_img.width,input_img.height,sizeof(double));
  img4 = (double **)get_img(input_img.width,input_img.height,sizeof(double));

  // /* Initialize the img arrays */
  // for ( i = 0; i < input_img.height; i++ )
  // for ( j = 0; j < input_img.width; j++ ) {
  //   img1[i][j] = 0;
  //   img2[i][j] = 0;
  // }


  /* copy green, red & blue component to double array */
  for ( i = 0; i < input_img.height; i++ )
```

```
61  for ( j = 0; j < input_img.width; j++ ) {
62    img1[i][j] = input_img.color[1][i][j];
63    imgr[i][j] = input_img.color[0][i][j];
64    imgb[i][j] = input_img.color[2][i][j];
65  }
66
67
68  /* Filter image with the F FIR Sharpening Filter */
69  for ( i = 2; i < input_img.height-2; i++ )
70  for ( j = 2; j < input_img.width-2; j++ ) {
71    // img2[i][j] = (img1[i][j-1] + img1[i][j] + img1[i][j+1])/3.0;
72    for ( ii = -2; ii <= 2; ii++ )
73    for ( jj = -2; jj <= 2; jj++ ) {
74      if (ii == 0 && jj == 0) {
75        img2[i][j] = img2[i][j] + (1+rho)*img1[i+ii][j+jj];
76        img3[i][j] = img3[i][j] + (1+rho)*imgr[i+ii][j+jj];
77        img4[i][j] = img4[i][j] + (1+rho)*imgb[i+ii][j+jj];
78      }
79      img2[i][j] = img2[i][j] - (rho*1/25)*img1[i+ii][j+jj];
80      img3[i][j] = img3[i][j] - (rho*1/25)*imgr[i+ii][j+jj];
81      img4[i][j] = img4[i][j] - (rho*1/25)*imgb[i+ii][j+jj];
82    }
83  }
84
85  /* Fill in boundary pixels */
86
87  // for ( i = 0; i < input_img.height; i++ ) {
88  //   img2[i][0] = 0;
89  //   img2[i][input_img.width-1] = 0;
90  // }
91
92  for ( i = 0; i < 2; i++ )
93  for ( j = 0; j < 2; j++ ) {
94    for ( ii = -1*i; ii <= 2; ii++ )
95    for ( jj = -1*i; jj <= 2; jj++ ) {
96      if (ii == 0 && jj == 0) {
97        img2[i][j] = img2[i][j] + (1+rho)*img1[i+ii][j+jj];
98        img3[i][j] = img3[i][j] + (1+rho)*imgr[i+ii][j+jj];
99        img4[i][j] = img4[i][j] + (1+rho)*imgb[i+ii][j+jj];
100     }
101     img2[i][j] = img2[i][j] - (rho*1/25)*img1[i+ii][j+jj];
102     img3[i][j] = img3[i][j] - (rho*1/25)*imgr[i+ii][j+jj];
103     img4[i][j] = img4[i][j] - (rho*1/25)*imgb[i+ii][j+jj];
104   }
105 }
106
107 for ( i = input_img.height-2; i < input_img.height; i++ )
108 for ( j = input_img.width-2; j < input_img.width; j++ ) {
109   for ( ii = -2; ii < input_img.height-i; ii++ )
110   for ( jj = -2; jj < input_img.width-j; jj++ ) {
111     if (ii == 0 && jj == 0) {
112       img2[i][j] = img2[i][j] + (1+rho)*img1[i+ii][j+jj];
113       img3[i][j] = img3[i][j] + (1+rho)*imgr[i+ii][j+jj];
114       img4[i][j] = img4[i][j] + (1+rho)*imgb[i+ii][j+jj];
115     }
116     img2[i][j] = img2[i][j] - (rho*1/25)*img1[i+ii][j+jj];
117     img3[i][j] = img3[i][j] - (rho*1/25)*imgr[i+ii][j+jj];
118     img4[i][j] = img4[i][j] - (rho*1/25)*imgb[i+ii][j+jj];
119   }
120 }
```

```
121
122   for ( i = 0; i < 2; i++ )
123   for ( j = input_img.width-2; j < input_img.width; j++ ) {
124     for ( ii = -1*i; ii <= 2; ii++ )
125     for ( jj = -2; jj < input_img.width-j; jj++ ) {
126       if (ii == 0 && jj == 0) {
127         img2[i][j] = img2[i][j] + (1+rho)*img1[i+ii][j+jj];
128         img3[i][j] = img3[i][j] + (1+rho)*imgr[i+ii][j+jj];
129         img4[i][j] = img4[i][j] + (1+rho)*imgb[i+ii][j+jj];
130       }
131       img2[i][j] = img2[i][j] - (rho*1/25)*img1[i+ii][j+jj];
132       img3[i][j] = img3[i][j] - (rho*1/25)*imgr[i+ii][j+jj];
133       img4[i][j] = img4[i][j] - (rho*1/25)*imgb[i+ii][j+jj];
134     }
135   }
136
137   for ( i = input_img.height-2; i < input_img.height; i++ )
138   for ( j = 0; j < 2; j++ ) {
139     for ( ii = -2; ii < input_img.height-i; ii++ )
140     for ( jj = -1*i; jj <= 2; jj++ ) {
141       if (ii == 0 && jj == 0) {
142         img2[i][j] = img2[i][j] + (1+rho)*img1[i+ii][j+jj];
143         img3[i][j] = img3[i][j] + (1+rho)*imgr[i+ii][j+jj];
144         img4[i][j] = img4[i][j] + (1+rho)*imgb[i+ii][j+jj];
145       }
146       img2[i][j] = img2[i][j] - (rho*1/25)*img1[i+ii][j+jj];
147       img3[i][j] = img3[i][j] - (rho*1/25)*imgr[i+ii][j+jj];
148       img4[i][j] = img4[i][j] - (rho*1/25)*imgb[i+ii][j+jj];
149     }
150   }
151
152
153   // /* Set seed for random noise generator */
154   // srandom2(1);
155
156   // /* Add noise to image */
157   // for ( i = 0; i < input_img.height; i++ )
158   // for ( j = 1; j < input_img.width-1; j++ ) {
159   //   img2[i][j] += 32*normal();
160   // }
161
162   /* set up structure for output achromatic image */
163   /* to allocate a full color image use type 'c' */
164   get_TIFF ( &green_img, input_img.height, input_img.width, 'g' );
165   get_TIFF ( &red_img, input_img.height, input_img.width, 'g' );
166   get_TIFF ( &blue_img, input_img.height, input_img.width, 'g' );
167
168   /* set up structure for output color image */
169   /* Note that the type is 'c' rather than 'g' */
170   get_TIFF ( &color_img, input_img.height, input_img.width, 'c' );
171
172   /* copy green, red & blue component to new images */
173   for ( i = 0; i < input_img.height; i++ )
174   for ( j = 0; j < input_img.width; j++ ) {
175     pixelg = (int32_t)img2[i][j];
176     pixelr = (int32_t)img3[i][j];
177     pixelb = (int32_t)img4[i][j];
178
179     if(pixelg>255) {
180       green_img.mono[i][j] = 255;
```

```
181     }
182     else {
183       if(pixelg<0) green_img.mono[i][j] = 0;
184       else green_img.mono[i][j] = pixelg;
185     }
186
187     if(pixelr>255) {
188       red_img.mono[i][j] = 255;
189     }
190     else {
191       if(pixelr<0) red_img.mono[i][j] = 0;
192       else red_img.mono[i][j] = pixelr;
193     }
194
195     if(pixelb>255) {
196       blue_img.mono[i][j] = 255;
197     }
198     else {
199       if(pixelb<0) blue_img.mono[i][j] = 0;
200       else blue_img.mono[i][j] = pixelb;
201     }
202   }
203
204   // /* Illustration: constructing a sample color image -- interchanging the red and
    green components from the input color image */
205   // for ( i = 0; i < input_img.height; i++ )
206   //     for ( j = 0; j < input_img.width; j++ ) {
207   //         color_img.color[0][i][j] = input_img.color[1][i][j];
208   //         color_img.color[1][i][j] = input_img.color[0][i][j];
209   //         color_img.color[2][i][j] = input_img.color[2][i][j];
210   //     }
211
212   /* Illustration: constructing a sample color image -- put 3 image (green,red,blue)
    into 1 image */
213   for ( i = 0; i < input_img.height; i++ )
214       for ( j = 0; j < input_img.width; j++ ) {
215           color_img.color[0][i][j] = red_img.mono[i][j];
216           color_img.color[1][i][j] = green_img.mono[i][j];
217           color_img.color[2][i][j] = blue_img.mono[i][j];
218       }
219
220   // /* open green image file */
221   // if ( ( fp = fopen ( "green.tif", "wb" ) ) == NULL ) {
222   //   fprintf ( stderr, "cannot open file green.tif\n");
223   //   exit ( 1 );
224   // }
225
226   // /* write green image */
227   // if ( write_TIFF ( fp, &green_img ) ) {
228   //   fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
229   //   exit ( 1 );
230   // }
231
232   // /* close green image file */
233   // fclose ( fp );
234
235
236   /* open color image file */
237   if ( ( fp = fopen ( "color.tif", "wb" ) ) == NULL ) {
238       fprintf ( stderr, "cannot open file color.tif\n");
```

```
239         exit ( 1 );
240     }
241
242     /* write color image */
243     if ( write_TIFF ( fp, &color_img ) ) {
244         fprintf ( stderr, "error writing TIFF file %s\n", argv[2] );
245         exit ( 1 );
246     }
247
248     /* close color image file */
249     fclose ( fp );
250
251     /* de-allocate space which was used for the images */
252     free_TIFF ( &(input_img) );
253     free_TIFF ( &(green_img) );
254     free_TIFF ( &(red_img) );
255     free_TIFF ( &(blue_img) );
256     free_TIFF ( &(color_img) );
257
258     free_img( (void**)img1 );
259     free_img( (void**)img2 );
260     free_img( (void**)img3 );
261     free_img( (void**)img4 );
262     free_img( (void**)imgr );
263     free_img( (void**)imgb );
264
265     return(0);
266 }
267
268 void error(char *name)
269 {
270     printf("usage:  %s  image.tiff \n\n",name);
271     printf("this program reads in a 24-bit color TIFF image.\n");
272     printf("It then horizontally filters the green component, adds noise,\n");
273     printf("and writes out the result as an 8-bit image\n");
274     printf("with the name 'green.tiff'.\n");
275     printf("It also generates an 8-bit color image,\n");
276     printf("that swaps red and green components from the input image");
277     exit(1);
278 }
279
280
```