// Gray to binary

unsigned char Gray;

//Perform conversions

Gray = fgetc(infptr);

do{

if (Gray > 127)

fputc(255,outfptr);

else

fputc(0,outfptr);

//Read next pixel to check if we have an End of File

Gray = fgetc(infptr);

} while (!feof(infptr));

// Negative

unsigned char Gray;

//Perform conversions

Gray = fgetc(infptr);

do{

fputc(255-Gray,outfptr);

//Read next pixel to check if we have an End of File

Gray = fgetc(infptr);

} while (!feof(infptr));

// Change output file to outfptrh

// Histogram

unsigned int histcount[256],Gray;

char string[10];

//Initialize histogram count to 0

for (int i=0;i<256;i++)

histcount[i] = 0;

//Perform conversions

Gray = fgetc(infptr);

do{

//Increment histogram for grayscale level

histcount[Gray]++;

//Read next pixel to check if we have an End of File

Gray = fgetc(infptr);

} while (!feof(infptr));

//Convert the histogram values from integers to string

for (int i=0;i<256;i++){

itoa (histcount[i],string,10);

fputs(string,outfptrh);

fputs("\n",outfptrh);

}

// RGB to Gray

unsigned char R,G,B,Gray;

char string[10];

//Perform conversions

R = fgetc(infptr);

do{

G = fgetc(infptr);

B = fgetc(infptr);

Gray = (R + G + B) / 3;

fputc(Gray,outfptr);

//Read next pixel to check if we have an End of File

R = fgetc(infptr);

} while (!feof(infptr));

// Mirror

unsigned char Gray;

unsigned char mat[512][512];

int ren, col;

//Perform conversions

for (ren = 0; ren < 512; ren++)

for (col = 0; col < 512; col++) {

Gray = fgetc(infptr);

mat[ren][col] = Gray;

}

for (ren = 0; ren < 512; ren++)

for (col = 0; col < 512; col++) {

//Invert columns

Gray = mat[ren][511-col];

fputc(Gray,outfptr);

}

// Reduction 50%

unsigned char Gray;

unsigned char mat[512][512];

int ren, col;

//Add \*.pgm Header to output file

fprintf(outfptr,"P5\n256 256\n255\n");

//Perform conversions

for (ren = 0; ren < 512; ren++)

for (col = 0; col < 512; col++) {

Gray = fgetc(infptr);

mat[ren][col] = Gray;

}

for (ren = 0; ren < 256; ren++)

for (col = 0; col < 256; col++) {

//Invert columns

Gray = mat[ren\*2][col\*2];

fputc(Gray,outfptr);

}

// Histogram Stretching

void negative()

{

unsigned char Gray;

//Matrix Variables

unsigned char mat[512][512];

int ren, col;

//Maximum and Minimum pixel values

int max = 0;

int min = 255;

//Correction Factor

float ScaleFactor;

//Add \*.pgm Header to output file

fprintf(outfptr,"P5\n512 512\n255\n");

//Read input image

for (ren = 0; ren < 512; ren++)

for (col = 0; col < 512; col++) {

//Read Pixel

Gray = fgetc(infptr);

//Store Pixel in Matrix

mat[ren][col] = Gray;

//Find lowest and highest pixel values in input image

if (Gray < min) min = Gray;

if (Gray > max) max = Gray;

}

printf("Maximum pixel value:%d\n", max);

printf("Minimum pixel value:%d\n", min);

//Correction Factor

ScaleFactor = 255.0/(max-min);

for (ren = 0; ren < 512; ren++)

for (col = 0; col < 512; col++) {

//Invert columns

Gray = mat[ren][col];

Gray = (Gray-min)\*ScaleFactor;

fputc(Gray,outfptr);

}

}

// Double size of an input image

void negative()

{

unsigned char Gray;

unsigned char mat[1024][1024];

int ren, col;

//Add \*.pgm Header to output file

fprintf(outfptr,"P5\n1024 1024\n255\n");

//Perform conversions

for (ren = 0; ren < 512; ren++)

for (col = 0; col < 512; col++) {

Gray = fgetc(infptr);

mat[ren\*2] [col\*2] = Gray;

mat[ren\*2+1][col\*2] = Gray;

mat[ren\*2] [col\*2+1] = Gray;

mat[ren\*2+1][col\*2+1] = Gray;

}

for (ren = 0; ren < 1024; ren++)

for (col = 0; col < 1024; col++) {

//Invert columns

Gray = mat[ren][col];

fputc(Gray,outfptr);

}

}

//------------------------------------------------------------------------------

//--------------------------Obtain convolved image------------------------------

//---------------by taking the one's complement of every pixel------------------

//------------------------------------------------------------------------------

void negative()

{

unsigned char Gray;

unsigned char mat [512][512];

unsigned char conv[512][512];

// Blur mask

// char mask[ 3][ 3] ={{ 1, 1, 1},

// { 1, 1, 1},

// { 1, 1, 1}};

// Sobel horizontal mask (edge detection)

// int mask[ 3][ 3] = {{ 1, 2, 1},

// { 0, 0, 0},

// {-1,-2,-1}};

// Sobel vertical mask (edge detection)

int mask[ 3][ 3] = {{-1, 0, 1},

{-2, 0, 2},

{-1, 0, 1}};

// Sharpening mask (second derivative)

// int mask[ 3][ 3] = {{ 0, 1, 0},

// { 1,-4, 1},

// { 0, 1, 0}};

// Sharpening mask (second derivative)

// int mask[ 3][ 3] = {{ 1, 1, 1},

// { 1,-8, 1},

// { 1, 1, 1}};

int col, row, value;

int dividefactor = 0;

// Calcula dividefactor

for (col = 0; col < 3; col++)

for (row = 0; row < 3; row++) {

dividefactor += mask[col][row];

}

printf ("dividefactor (antes) = %d \n",dividefactor);

if (dividefactor == 0)

dividefactor = 1;

printf ("dividefactor (despues) = %d \n",dividefactor);

//Add \*.pgm Header to output file

fprintf(outfptr,"P5\n512 512\n255\n");

//Read input image and store en matrix

for (col = 0; col < MRows; col++)

for (row = 0; row < NCols; row++) {

Gray = fgetc(infptr);

mat[col][row] = Gray;

}

// Clear convolution matrix

for (col = 0; col < MRows; col++)

for (row = 0; row < NCols; row++)

conv[col][row] = 0;

//Convolve image with mask

for (col = 1; col < MRows-1; col++)

for (row = 1; row < NCols-1; row++) {

value = ((mask[0][0] \* mat[col-1][row-1] +

mask[0][1] \* mat[col ][row-1] +

mask[0][2] \* mat[col+1][row-1] +

mask[1][0] \* mat[col-1][row ] +

mask[1][1] \* mat[col ][row ] +

mask[1][2] \* mat[col+1][row ] +

mask[2][0] \* mat[col-1][row+1] +

mask[2][1] \* mat[col ][row+1] +

mask[2][2] \* mat[col+1][row+1]) / dividefactor);

conv[col][row] = value;

if (value < 0) conv[col][row] = 0;

else if (value > 255) conv[col][row] = 255;

else conv[col][row] = value;

}

// printf("Fisrt convolved pixel = %d \n",conv[1][1]);

//Save convolved image

for (col = 0; col < MRows; col++)

for (row = 0; row < NCols; row++) {

Gray = conv[col][row];

fputc(Gray,outfptr);

}

}