#include <iostream>

#include<fstream>

#include<vector>

#include<cmath>

#include<cstring>

using namespace std;

struct Image {

char ImageFileName[100];

vector<vector<int>> ImageData;

int cols, rows, maxGray;

vector<char> comment;

bool imageLoaded;

bool imageModified;

void changeBrightness(double factor) {

for (int r = 0; r < rows; r++)

for (int c = 0; c < cols; c++) {

ImageData[r][c] \*= factor;

if (ImageData[r][c] > maxGray)

ImageData[r][c] = maxGray;

}

}

void LinearContrastStreatching()

{

int min = maxGray, max = 0;

for (auto i = 0; i<ImageData.size(); i++)

{

for (int j = 0; j < ImageData[0].size(); j++)

{

if (ImageData[i][j] > max)

max = ImageData[i][j];

if (ImageData[i][j] < min)

min = ImageData[i][j];

}

}

for (auto i = 0; i < ImageData.size(); i++)

{

for (int j = 0; j < ImageData[0].size(); j++)

{

ImageData[i][j] = (ImageData[i][j] - min) \* 255 / (max - min);

}

}

}

void sharpenImage()

{

vector<vector<float>> kernel = {

{-1,-1,-1},

{-1,5,-1},

{-1,-1,-1}

};

applyFilter(kernel);

}

void binaryImage(float limit)

{

for (auto i = 0; i < ImageData.size(); i++)

{

for (int j = 0; j < ImageData[0].size(); j++)

{

if (ImageData[i][j] >= limit \* maxGray)

ImageData[i][j] = maxGray;

else

ImageData[i][j] = 0;

}

}

}

void resizeImage(float xfactor, float yfactor)

{

int new\_cols = cols \* xfactor, new\_rows = rows \* yfactor;

vector <vector <int>> resizedImage(new\_rows, vector<int>(new\_cols) );

for (int i = 0; i < new\_rows; i++)

{

for (int j = 0; j < new\_cols; j++)

{

resizedImage[i][j] = ImageData[int(i / yfactor)][int(j / xfactor)];

}

}

rows = new\_rows;

cols = new\_cols;

ImageData = resizedImage;

resizedImage.clear();

}

void rotateImage(int degree)

{

double radian = degree \* 3.1416 /180.0;

int x1 = 0;

int y1 = 0;

int x2 = rows - 1;

int y2 = 0;

int x3 = 0;

int y3 = cols - 1;

int x4 = rows- 1;

int y4 = cols- 1;

int x1\_rotated = round(x1 \* cos(radian) - y1 \* sin(radian));

int y1\_rotated = round(x1 \* sin(radian) + y1 \* cos(radian));

int x2\_rotated = round(x2 \* cos(radian) - y2 \* sin(radian));

int y2\_rotated = round(x2 \* sin(radian) + y2 \* cos(radian));

int x3\_rotated = round(x3 \* cos(radian) - y3 \* sin(radian));

int y3\_rotated = round(x3 \* sin(radian) + y3 \* cos(radian));

int x4\_rotated = round(x4 \* cos(radian) - y4 \* sin(radian));

int y4\_rotated = round(x4 \* sin(radian) + y4 \* cos(radian));

int rotatedRows = max({ x1\_rotated, x2\_rotated, x3\_rotated, x4\_rotated }) - min({ x1\_rotated, x2\_rotated, x3\_rotated, x4\_rotated }) + 1;

int rotatedCols = max({ y1\_rotated, y2\_rotated, y3\_rotated, y4\_rotated }) - min({ y1\_rotated, y2\_rotated, y3\_rotated, y4\_rotated }) + 1;

vector<vector<int>> rotated(rotatedRows, vector<int>(rotatedCols, 0));

int centerX = rotatedRows / 2;

int centerY = rotatedCols / 2;

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

for (int j = 0; j < rotatedCols; j++) {

int x\_trans = i - centerX;

int y\_trans = j - centerY;

int x\_rotated = round(x\_trans \* cos(radian) - y\_trans \* sin(radian));

int y\_rotated = round(x\_trans \* sin(radian) + y\_trans \* cos(radian));

int x\_final = x\_rotated + centerX;

int y\_final = y\_rotated + centerY;

if (x\_final >= 0 && x\_final < rows && y\_final >= 0 && y\_final < cols) {

rotated[i][j] = ImageData[x\_final][y\_final];

}

}

}

rows = rotatedRows;

cols = rotatedCols;

ImageData = rotated;

}

int loadImage(char ImageName[]) {

ifstream FCIN(ImageName);

if (!FCIN.is\_open())

return -1;

char MagicNumber[5];

char Comment[100];

FCIN.getline(MagicNumber, 4);

FCIN.getline(Comment, 100);

FCIN >> cols >> rows >> maxGray;

ImageData.clear();

ImageData.resize(rows, vector<int>(cols, 0));

for (int r = 0; r < rows; r++)

for (int c = 0; c < cols; c++)

FCIN >> ImageData[r][c];

if (FCIN.fail())

return -2;

FCIN.close();

imageLoaded = true;

imageModified = false;

strcpy\_s(ImageFileName, sizeof(ImageFileName), ImageName);

return 0;

}

int saveImage(char ImageName[]) {

ofstream FCOUT(ImageName);

if (!FCOUT.is\_open())

return -1;

FCOUT << "P2\n# This is a comment\n"

<< cols << " " << rows << endl << maxGray << endl;

for (int r = 0; r < rows; r++) {

for (int c = 0; c < cols; c++)

FCOUT << ImageData[r][c] << " ";

FCOUT << endl;

}

FCOUT.close();

imageModified = false;

return 0;

}

void verticalFlipImage() {

for (int r = 0; r < rows / 2; r++)

swap(ImageData[r],ImageData[rows-r-1]);

}

void horizontalFlipImage() {

for (int r = 0; r < rows; r++)

for (int c = 0; c < cols/2; c++)

{

swap(ImageData[r][c], ImageData[r][cols - c - 1]);

}

}

void crop(int startx, int starty, int endx, int endy)

{

int difx = endx - startx;

int dify = endy - starty;

vector < vector <int>> croped(difx, vector<int>(dify));

for (int i = startx; i < endx; i++)

{

for (int j = starty; j < endy; j++)

{

if (i < rows && j < cols)

{

croped[i - startx][j - starty] = ImageData[i][j];

}

}

}

rows = difx;

cols = dify;

ImageData = croped;

}

void combineImages(Image image2, int option)

{

if (option != 0 && option != 1)

{

cout << "enter a valid option next time.\n";

return;

}

if (option == 0)

{

int new\_rows = max(rows, image2.rows);

int new\_cols = cols + image2.cols;

vector < vector <int>> combined(new\_rows, vector<int>(new\_cols));

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

{

for (int j = 0; j < cols; j++)

{

combined[i][j] = ImageData[i][j];

}

}

for (int i = 0; i < image2.rows; i++)

{

for (int j = cols; j < new\_cols; j++)

{

if (i < image2.rows)

{

combined[i][j] = image2.ImageData[i][j-cols];

}

}

}

cols = new\_cols;

rows = new\_rows;

ImageData = combined;

}

else

{

int new\_cols = max(cols, image2.cols);

int new\_rows = rows + image2.rows;

vector < vector <int>> combined(new\_rows, vector<int>(new\_cols));

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

{

for (int j = 0; j < cols; j++)

{

combined[i][j] = ImageData[i][j];

}

}

for (int i = rows; i < new\_rows; i++)

{

for (int j = 0; j < image2.cols; j++)

{

if (j < image2.cols)

{

combined[i][j] = image2.ImageData[i-rows][j];

}

}

}

cols = new\_cols;

rows = new\_rows;

ImageData = combined;

}

}

void bubbleSort(vector<int>& vec) {

int n = vec.size();

bool swapped;

do {

swapped = false;

for (int i = 1; i < n; ++i) {

if (vec[i - 1] > vec[i]) {

swap(vec[i - 1], vec[i]);

swapped = true;

}

}

--n;

} while (swapped);

}

void applyMeanMedian(int choice)

{

if (choice != 0 && choice != 1)

{

cout << "enter a valid option next time.\n";

return;

}

vector< vector<int>> filtered(rows, vector<int>(cols));

int kernal\_size = 3;

if (choice == 0)

{

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

{

for (int j = 0; j < cols; j++)

{

int sum = 0;

int vals = 0;

for (int k = -kernal\_size / 2; k <= kernal\_size / 2; k++)

{

if (i + k < 0 || i + k >= rows)

continue;

for (int l = -kernal\_size / 2; l <= kernal\_size / 2; l++)

{

if (j + l < 0 || j + l >= cols)

continue;

sum += ImageData[i][j];

vals++;

}

}

filtered[i][j] = sum/vals;

}

}

}

else

{

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

{

for (int j = 0; j < cols; j++)

{

vector<int> vals(9);

int ind = 0;

for (int k = -kernal\_size / 2; k <= kernal\_size / 2; k++)

{

if (i + k < 0 || i + k >= rows)

continue;

for (int l = -kernal\_size / 2; l <= kernal\_size / 2; l++)

{

if (j + l < 0 || j + l >= cols)

continue;

vals[ind] = ImageData[i][j];

ind++;

}

}

bubbleSort(vals);

size\_t size = vals.size();

int med;

if (size % 2 == 0) {

med = (vals[size / 2 - 1] + vals[size / 2]) / 2;

}

else {

med = vals[size / 2];

}

filtered[i][j] = med;

}

}

}

ImageData = filtered;

}

void applyLinearFilter(char filename[])

{

ifstream file(filename);

if (!file.is\_open()) {

cerr << "Error: Unable to open file " << filename << endl;

return;

}

vector<vector<float>> kernel;

int r\_size, c\_size;

file >> r\_size >> c\_size;

kernel.resize(r\_size, vector<float>(c\_size));

for (int i = 0; i < r\_size; ++i) {

for (int j = 0; j < c\_size; ++j) {

if (!(file >> kernel[i][j])) {

cerr << "Error: Unable to read matrix element at position (" << i << ", " << j << ")" << endl;

return;

}

}

}

applyFilter(kernel);

}

void translateImage(int x, int y)

{

rows -= y;

cols -= x;

}

void applyFilter(vector< vector<float>> kernel){

vector < vector<int>> transformed(rows, vector <int>(cols));

for (int i = 0; i < ImageData.size(); i++) {

for (int j = 0; j < ImageData[i].size(); j++) {

float sum = 0;

for (int k = 0; k < kernel.size(); k++) {

for (int l = 0; l < kernel[0].size(); l++) {

int ii = i + k - kernel.size()/ 2;

int jj = j + l - kernel[0].size()/ 2;

if (ii >= 0 && ii < rows && jj >= 0 && jj < cols) {

sum += ImageData[ii][jj] \* kernel[k][l];

}

}

}

transformed[i][j] = sum;

}

}

ImageData = transformed;

}

};

struct Menu {

vector<string> menuItems;

Menu(char menuFile[]) {

loadMenu(menuFile);

}

int loadMenu(char menuFile[]) {

ifstream IN;

IN.open(menuFile);

if (!IN.is\_open())

return -1;

char menuItem[100], TotalItems[10];

int Choices;

IN.getline(TotalItems, 8);

Choices = atoi(TotalItems);

for (int i = 1; i <= Choices; i++) {

IN.getline(menuItem, 99);

menuItems.push\_back(menuItem);

}

IN.close();

return Choices;

}

int presentMenu() {

int userChoice;

int totalChoices = menuItems.size();

do {

int k = 1;

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

if (menuItems[i][0] != '\*') {

cout << k << "\t" << menuItems[i] << endl;

k++;

}

}

cout << " Enter Your Choice (1 - " << k - 1 << " ) ";

cin >> userChoice;

} while (userChoice < 1 || userChoice > totalChoices);

return userChoice;

}

};

int main() {

char MenuFile[] = "MainMenu.txt";

Image images[2];

int activeImage = 0;

int errorCode = 0;

int userChoice;

// int TotalChoices = loadMenu("MainMenu.txt");

int totalChoices;

Menu menu(MenuFile);

totalChoices = menu.menuItems.size();

do {

userChoice = menu.presentMenu();

if (1 == userChoice)

{

char ImageFileName[100];

cout << "Specify File Name ";

cin >> ImageFileName;

errorCode = images[activeImage].loadImage(ImageFileName);

if (errorCode == 0) {

cout << "File Loaded Successfully " << endl;

}

else {

cout << "Load Error: Code " << errorCode << endl;

}

}

else if (2 == userChoice) {

char ImageFileName[100];

cout << "Specify File Name ";

cin >> ImageFileName;

errorCode = images[activeImage].saveImage(ImageFileName);

if (errorCode == 0) {

cout << "File Saved as " << ImageFileName << endl;

string command = "start /wait \"\" \"D:\\APPS\_DATA\\IrfanView\\i\_view64.exe\" \"" + std::string(ImageFileName) + "\"";

system(command.c\_str());

}

else {

cout << "Save Error: Code " << errorCode << endl;

}

}

else if (3 == userChoice) {

float fac;

cout << "Enter brightess factor: ";

cin >> fac;

images[activeImage].changeBrightness(fac);

}

else if (4 == userChoice) {

images[activeImage].LinearContrastStreatching();

cout << "You need to save the changes " << endl;

}

else if (5 == userChoice) {

images[activeImage].sharpenImage();

cout << "You need to save the changes " << endl;

}

else if (6 == userChoice) {

float fac;

cout << "Enter the threshold value between 0 and 1: ";

cin >> fac;

images[activeImage].binaryImage(fac);

cout << "You need to save the changes " << endl;

}

else if (7 == userChoice) {

float facx, facy;

cout << "Enter the resize factor for x and y: ";

cin >> facx >> facy;

images[activeImage].resizeImage(facx, facy);

cout << "You need to save the changes " << endl;

}

else if (8 == userChoice) {

int degree;

cout << "Enter the rotation angle (put give negative value for anti-clockwise rotation): ";

cin >> degree;

images[activeImage].rotateImage(degree);

cout << "You need to save the changes " << endl;

}

else if (9 == userChoice) {

images[activeImage].horizontalFlipImage();

cout << "You need to save the changes " << endl;

}

else if (10 == userChoice) {

images[activeImage].verticalFlipImage();

cout << "You need to save the changes " << endl;

}

else if (11 == userChoice) {

int x1,x2,y1,y2;

cout << "Enter values of x1,y1 and x2,y2 in the repective order: ";

cin >> x1 >> y1 >> x2 >> y2;

images[activeImage].crop(x1,y1,x2,y2);

cout << "You need to save the changes " << endl;

}

else if (12 == userChoice) {

char ImageFileName2[100];

cout << "Specify 2nd File Name ";

cin >> ImageFileName2;

errorCode = images[(activeImage + 1)%2].loadImage(ImageFileName2);

if (errorCode == 0) {

cout << "File Loaded Successfully " << endl;

int option;

cout << "0 Side By Side\n1 Top to Bottom\n";

cin >> option;

images[activeImage].combineImages(images[(activeImage + 1) % 2], option);

cout << "You need to save the changes " << endl;

}

else {

cout << "Load Error: Code " << errorCode << endl;

}

}

else if (13 == userChoice) {

int x1;

cout << "0 Mean\n1 Median\n";

cin >> x1;

images[activeImage].applyMeanMedian(x1);

cout << "You need to save the changes " << endl;

}

else if (14 == userChoice)

{

char filename[100];

cout << "Enter file name: ";

cin >> filename;

images[activeImage].applyLinearFilter(filename);

cout << "You need to save the changes " << endl;

}

else if (15 == userChoice)

{

int rows, cols;

cout << "Enter the number of rows: ";

cin >> rows;

cout << "Enter the number of columns: ";

cin >> cols;

vector<vector<float>> matrix(rows, vector<float>(cols));

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

for (int j = 0; j < cols; j++) {

cout << "Enter element (" << i + 1 << ", " << j + 1 << "): ";

cin >> matrix[i][j];

}

}

images[activeImage].applyFilter(matrix);

cout << "You need to save the changes " << endl;

}

else if (16 == userChoice)

{

// Roberts Filter

vector<vector<float>> kernel = {

{1, 0},

{0, -1},

};

images[activeImage].applyFilter(kernel);

cout << "You need to save the changes " << endl;

}

else if (17 == userChoice)

{

// Laplacian Filter

vector<vector<float>> kernel = {

{0, 1, 0},

{1, -4, 1},

{0, 1, 0 }

};

images[activeImage].applyFilter(kernel);

cout << "You need to save the changes " << endl;

}

else if (18 == userChoice)

{

int x, y;

cout << "Enter values for x and y: ";

cin >> x >> y;

images[activeImage].translateImage(x,y);

cout << "You need to save the changes " << endl;

}

else if (19 == userChoice)

{

float fac;

cout << "Enter the scaling factor: ";

cin >> fac;

images[activeImage].resizeImage(fac, fac);

cout << "You need to save the changes " << endl;

}

} while (userChoice != totalChoices);

return 0;

}