**Laberinto**

**Definición y análisis del problema**

Realizar un programa que al ingresar una imagen permita mostrar el negativo, ampliarla, reducirla y ecualizarla.

**Salida:** Negativo, imagen ampliada, imagen reducida, imagen ecualizada.

**Entrada** Imagen original.

**Proceso: Negativo:** a 255 restarle el valor de cada byte en la imagen.

**Ecualizado:** encontrar la ecuación de la recta del histograma y modificarla de tal manera que la imagen deje de verse blanca u obscura.

**Ampliación:** multiplicar por 2 el número de filas y columnas y en los lugares sobrantes poner una copia de la columna o renglón anterior

**Reducción:** dividir entre 2 el número de filas y columnas, eliminando de forma alternada filas y columnas

**Planeación de la solución**

Imagen

Imagen

Imagen

Reducción

Leer imagen

Imagen

Imagen

Reducir bytes a la mitad

Ampliación

Pedir imagen

Negativo

Ecualización

Duplicar bytes

Leer imagen

Ajustar pendiente

Restar cada byte a 255

Imagen ampliada

Imagen reducida

Imagen negativa

Imagen equalizada

Guardar imagen

**Principal**

#include "stdafx.h"

#include <iomanip>

#include <iostream>

#include <string>

#include <string.h>

#include <stdlib.h>

#include <new.h>

#include <fstream>

using namespace std;

using std::ios;

class Imagen

{

//Atributos de la clase

private:

unsigned int ancho, alto,anchom,altom;

char encabezado[16], \*\*dimimagen=NULL,\*\*imagenmodi= NULL;

public:

//Constructor

Imagen();

unsigned int getancho();

unsigned int getalto();

char getencabezado();

//Metodos de la clase

void LeerImagen();

void GuardarImagen();

void GuardarImagenModificada();

void Negativo();

void Ecualiza();

void Aumenta();

void Reduce();

};

void Introduccion(void);

void Menu(void);

int main()

{

int opc;

Imagen foto; //Nombre del objeto

Introduccion();

foto.LeerImagen();

do

{

Menu();

cin >> opc;

cin.ignore();

system("cls");

switch (opc)

{

case 1:

{

foto.Negativo();

system("cls");

break;

}

case 2:

{

foto.Ecualiza();

system("cls");

break;

}

case 3:

{

foto.Aumenta();

system("cls");

break;

}

case 4:

{

foto.Reduce();

system("cls");

break;

}

case 5:

{

cout << "Adios" << endl;

break;

}

default:

{

cout << "Error. Vuelva a intentarlo" << endl;

break;

}

}

} while (opc != 5);

system("pause");

return 0;

}

void Introduccion()

{

cout << "Nombre de quienes elaboraron este proyecto" << endl

<< "Aranda Mejia Brian Antonio" << endl

<< "Narvaez Gol Juan" << endl;

system("pause");

system("cls");

cout << "El objetivo del programa es: Crear un programa el cual realice procesamiento digital de imágenes en formato PGM.\n"

<< "El cual debera ser capaz de sacar el negativo de una imagen PGM, la ecualización lineal de una imagen PGM,\n"

<< "reduccion a la mitad de una imagen PGM y ampliacion a la mitad de una imagen PGM." << endl;

system("pause");

system("cls");

}

void Menu()

{

cout << "Menu" << endl

<< "Que proceso desea realiazar sobre la imagen?" << endl

<< "1. Obtener el negativo de una imagen" << endl

<< "2. Ecualizacion lineal de una imagen" << endl

<< "3. Agrandar la imagen" << endl

<< "4. Reduccion de una imagen" << endl

<< "5. Salir" << endl;

}

Imagen::Imagen() //Constructor del Objeto Imagen

{

ancho = 0;

alto = 0;

}

unsigned int Imagen::getancho() //Metodo get de la clase del atributo ancho

{

return ancho;

}

unsigned int Imagen::getalto() //Metodo get de la clase del atributo alto

{

return alto;

}

char Imagen::getencabezado() //Metodo get de la clase del atributo encabezado

{

return encabezado[16];

}

void Imagen::LeerImagen() //En este metodo de la clase imagen se puede leer la imagen que se desea modificar

{

ifstream imagen;

string archivoimagen;

char pix;

unsigned int x = 0, i, j;

do

{

cout << "Que imagen quieres leer?" << endl;

getline(cin, archivoimagen);

system("cls");

imagen.open(archivoimagen.c\_str(), ios::in | ios::binary);

if (imagen.fail())

{

cout << "No se pudo abrir la imagen. Vuelva a intentarlo\n" << endl;

system("pause");

}

} while (imagen.fail()); //Ciclo que se repite si la imagen no existe o no se pudo abrir

imagen.get(encabezado, 16);

encabezado[15] = '\0';

sscanf\_s(encabezado, "P5 %d %d 255 ", &ancho, &alto); //Lee el encabezado de la imagen para poder sacar sus dimensiones

dimimagen = new char \*[alto]; //Se crea el arreglo dinamico

for (i = 0; i <alto; i++)

{

dimimagen[i] = new char[ancho];

}

while (x<alto)

{

for (j = 0; j < ancho; j++)

{

imagen.get(pix);

dimimagen[x][j] = pix;

}

x++;

}

imagen.close();

}

void Imagen::GuardarImagen() //En este metodo de la clase imagen se puede guardar la imagen que ya se modifico

{

unsigned int i, j;

ofstream imagen;

string nuevoarchivo;

cout << "Nombre del archivo que deseas guardar:" << endl;

getline(cin, nuevoarchivo);

imagen.open(nuevoarchivo.c\_str(), ios::out | ios::binary);

imagen << encabezado;

for (i = 0; i < alto; i++)

{

for (j = 0; j < ancho; j++)

{

imagen.put(dimimagen[i][j]);

}

}

imagen.close();

for (i = 0; i < alto; i++) //Borra el arreglo dinamico para que no se este almacenado cada que se ejecuta el programa

{

delete[] dimimagen[i];

}

delete[] dimimagen;

}

void Imagen::Negativo() //En este metodo se puede sacar el negativo de la imagen

{

unsigned int i, j;

char pix, nuevopix;

for (i = 0; i < alto; i++)

{

for (j = 0; j < ancho; j++)

{

pix = dimimagen[i][j];

nuevopix= (pix \* (-1)) + 255;

dimimagen[i][j] = nuevopix;

}

}

GuardarImagen();

}

void Imagen::Ecualiza() //En este metodo se ecualiza la imagen

{

unsigned int i, j,m;

char pix, recta;

int mayor = 0, menor = 255;

for (i = 0; i < alto; i++)

{

for (j = 0; j < ancho; j++)

{

pix = dimimagen[i][j];

if (mayor<pix)

{

mayor = pix;

}

if (menor>pix)

{

menor = pix;

}

m = (255) / (mayor - menor); //Calculo de la pendiente

recta = (m\*pix) - (m\*menor); //Se saca el valor de y

dimimagen[i][j] = recta;

}

}

GuardarImagen();

}

void Imagen::Aumenta() //En este metodo se aumenta al doble la imagen

{

unsigned int x = 0, y = 0, i, j;

char pix,aux;

altom = alto \* 2;

anchom = ancho \* 2;

imagenmodi = new char \*[altom];

for (i = 0; i < altom; i++)

{

imagenmodi[i] = new char[anchom];

}

for (i = 0; i < altom; i++)

{

for (j = 0; j < anchom; j++)

{

pix = dimimagen[x][y];

aux = dimimagen[x][y];

imagenmodi[i][j] = aux+pix;

}

}

GuardarImagenModificada();

}

void Imagen::Reduce() //En este metodo se reduce al doble la imagen

{

unsigned int x=0, y=0, i, j;

char pix;

altom = alto / 2;

anchom = ancho / 2;

imagenmodi = new char \*[altom];

for (i = 0; i < altom; i++)

{

imagenmodi[i] = new char[anchom];

}

for (i = 0; i < altom; i++)

{

for (j = 0; j < anchom; j++)

{

pix = dimimagen[x][y+2];

imagenmodi[i][j] = pix;

}

x++;

y = 0;

}

GuardarImagenModificada();

}

void Imagen::GuardarImagenModificada() //Se creo un metodo de guardar aparte, el cual nos sirve especificamente para imagenes que se quieren agrandar o reducir

{

unsigned int i, j;

ofstream imagen;

string nuevoarchivo;

cout << "Nombre del archivo que deseas guardar:" << endl;

getline(cin, nuevoarchivo);

imagen.open(nuevoarchivo.c\_str(), ios::out | ios::binary);

imagen <<encabezado;

for (i = 0; i < altom; i++)

{

for (j = 0; j < anchom; j++)

{

imagen.put(imagenmodi[i][j]);

}

}

imagen.close();

for (i = 0; i < altom; i++)

{

delete[] imagenmodi[i];

}

delete[] imagenmodi;

}

**Pruebas**

































