МИНИСТЕРСТВО ВЫСШЕГО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное образовательное учреждение высшего образования

Санкт-Петербургский национальный исследовательский университет информационных

технологий, механики и оптики

Мегафакультет трансляционных информационных технологий

Факультет информационных технологий и программирования

**Отчет по лабораторной работе №1**

**По дисциплине «Компьютерная геометрия и графика»**

**Изучение простых преобразований изображений**

**Выполнил студент группы №M3234:**

***Ота Никита Терентьевич***

**Преподаватель:**

***Скаков Павел Сергеевич***

**САНКТ-ПЕТЕРБУРГ  
2020**

Цель работы:

Изучить алгоритмы и реализовать программу выполняющую простые преобразования серых и цветных изображений в формате PNM.

**Описание работы:**

Программа должна поддерживать серые и цветные изображения (варианты PNM P5 и P6), самостоятельно определяя формат по содержимому.

Аргументы программе передаются через командную строку:  
lab#.exe <имя\_входного\_файла> <имя\_выходного\_файла> <преобразование>, где <преобразование>:  
0 — инверсия,  
1 - зеркальное отражение по горизонтали,  
2 - зеркальное отражение по вертикали,  
3 - поворот на 90 градусов по часовой стрелке,  
4 - поворот на 90 градусов против часовой стрелки.

Программа должна быть написана на C/C++ и не использовать внешние библиотеки.

**Теоретическая часть:**

В начале файла изображения содержится метка, какое изображение там хранится: если P5 – монохромное, если P6 – цветное (RGB).

На следующей строке содержится ширина и высота изображения, представленные целым десятичным числом и разделённые через пробельный символ.

В третьей строке содержится максимально возможное значение яркости (наибольшее из них – 255).

Начитая с четвёртой строки идут данные изображения в виде байт-кода (к примеру, если содердится символ перевода строки ‘\n’, то этот байт расценивается как число 10). В зависимости от того, какой тип изображения, значения байтов различаются. Для P5 – каждый байт представляет яркость текущего пикселя, а для P6 – яркость одного из цветов: красного, зелёного и синего, и при этом эти байты идут в порядке RGB.

**Первая подзадача: инверсия.**Монохромные и триколорные (RGB) картинки инвертируются одинаковым способом: нужно из наибольшего значения вычесть значения в пикселе. В данном задании наибольшим значением для пикселя является 255.

**Третья подзадача: зеркальное отражение по вертикали.**   
Выполняется обменом местами строк из таблицы.

**Вторая подзадача: зеркальное отражение по горизонтали.**Выполняется так же, как и третья подзадача, но вместо строк нужно менять положения столбцов.

**Четвёртая подзадача: поворот на 90 градусов по часовой стрелке.**Первая строка отображается в последний столбец, вторая на предпоследний и так далее.

**Пятая подзадача: поворот на 90 градусоа против часовой стрелки.**Эта операция обратна предыдущей: последний столбец отображается в первую строку и так далее.

**На что нужно было обратить внимание.**В процессе выполнения программы могли возникать ошибки, которые нарушали бы правильность выполнения. Их нужно было обработать.

**Практическая часть**

Язык, на котором написан код, выполняющий задание: С++.

Исходный код:

#include <fstream>

#include <string>

#include <stdio.h>

#include <vector>

// int const BUF\_LEN = 5;

typedef unsigned char uchar;

int const BYTE\_MAX = 1 << 8 - 1;

namespace reader {

int const BUF\_LEN = 1 << 8, RGB\_BYTES = 3;

uchar \*buffer = new uchar[BUF\_LEN];

char p;

int type;

int height, width;

unsigned int max\_code;

std::ifstream in;

int curr\_pos = 0, end\_pos = 0, read = 0;

char get\_next() {

if (curr\_pos == end\_pos) {

in.read((char \*) buffer, BUF\_LEN);

end\_pos = in.gcount();

if (end\_pos == 0) {

throw std::runtime\_error("An input error occurred while reading file");

}

curr\_pos = 0;

}

++read;

return buffer[curr\_pos++];

}

}

struct pixel {

std::vector<uchar> info;

pixel() {

try {

if (reader::type == 5) {

info.resize(1);

} else {

info.resize(3);

}

} catch (...) {

throw std::runtime\_error("No free memory to allocate pixel");

}

}

};

namespace reader {

std::vector<std::vector<pixel>> data;

void get\_next(pixel &to) {

if (type == 5) {

to.info[0] = get\_next();

} else {

// for (int &i : to.info) {

// i = get\_next();

// }

for (int i = 0; i < RGB\_BYTES; ++i) {

to.info[i] = get\_next();

}

}

}

}

namespace writer {

int height, width;

std::ofstream out;

std::vector<std::vector<pixel>> data;

void set\_data(int a, int b) {

height = a;

width = b;

try {

data.resize(a, std::vector<pixel>(b));

} catch (...) {

throw std::runtime\_error("No free memory to allocate writer::data");

}

}

void write\_meta() {

try {

out << reader::p << reader::type << '\n';

out << width << ' ' << height << '\n';

out << reader::max\_code << '\n';

} catch (...) {

throw std::runtime\_error("An error occurred while writing meta info");

}

}

void write\_data() {

for (auto &i : data) {

for (auto &j : i) {

for (uchar k : j.info) {

try {

out.write((char \*) &k, 1);

} catch (...) {

throw std::runtime\_error("An error occurred while writing data");

}

}

}

}

}

}

namespace edit {

void inverse() {

printf("inverse\n");

writer::set\_data(reader::height, reader::width);

for (int i = 0; i < reader::height; ++i) {

for (int j = 0; j < reader::width; ++j) {

for (int k = 0; k < reader::data[i][j].info.size(); ++k) {

writer::data[i][j].info[k] = reader::max\_code - reader::data[i][j].info[k];

}

}

}

}

namespace mirror {

void vertical() {

printf("vertical mirror\n");

writer::set\_data(reader::height, reader::width);

// for (int i = 0; i < reader::height; ++i) {

// writer::data[i] = reader::data[reader::height - 1 - i];

// }

for (int i = 0; i < writer::height; ++i) {

for (int j = 0; j < writer::width; ++j) {

writer::data[i][j] = reader::data[reader::height - i - 1][j];

}

}

}

void horisontal() {

printf("horizontal mirror\n");

writer::set\_data(reader::height, reader::width);

for (int i = 0; i < writer::height; ++i) {

for (int j = 0; j < writer::width; ++j) {

writer::data[i][j] = reader::data[i][reader::width - 1 - j];

}

}

}

}

namespace rotate {

/\* 1 2 3 4 1 1 1 <- 2 1 1 2 <- 1 1

4 5 6 -> 5 2 2 1 <- 2 2 2 2 <- 1 2

6 3 3 1 <- 2 3 3 2 <- 1 3

\*/

void clockwise() {

printf("clockwise rotate\n");

writer::set\_data(reader::width, reader::height);

for (int i = 0; i < writer::height; ++i) {

for (int j = 0; j < writer::width; ++j) {

writer::data[i][j] = reader::data[reader::height - j - 1][i];

}

}

}

/\* 1 2 3 3 6 1 1 <- 1 3 1 2 <- 2 3

4 5 6 -> 2 5 2 1 <- 1 2 2 2 <- 2 2

1 4 3 1 <- 1 1 3 2 <- 2 1

\*/

void counterclockwise() {

printf("counterclockwise\n");

writer::set\_data(reader::width, reader::height);

for (int i = 0; i < writer::height; ++i) {

for (int j = 0; j < writer::width; ++j) {

writer::data[i][j] = reader::data[j][reader::width - 1 - i];

}

}

}

}

}

void run(int argc, char \*argv[], bool debug) {

if (argc != 4) {

throw std::runtime\_error("Expected 3 arguments: input file, output file, command");

}

reader::in.open(argv[1], std::ios::in | std::ios::binary);

if (!reader::in.is\_open()) {

throw std::runtime\_error("Input file can't be opened");

}

{

using namespace reader;

in >> p >> type;

printf("%c %d\n", p, type);

if (p != 'P' || (type > 6 || type < 5)) {

throw std::runtime\_error("Expected P5 or P6 in input file");

}

in >> width >> height >> max\_code;

get\_next();

try {

data.resize(height, std::vector<pixel>(width));

} catch (...) {

throw std::runtime\_error("No free memory to allocate reader::data");

}

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

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

get\_next(data[i][j]);

}

}

if (debug) {

std::string s = argv[1];

std::ofstream fout(s + "decoded\_from\_input.pnm");

fout << p << type - 3 << '\n' << width << ' ' << height << '\n' << max\_code << '\n';

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

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

for (auto &k : data[i][j].info) {

fout << k << ' ';

}

}

}

fout.close();

}

}

printf("h = %d, w = %d, max = %d\n", reader::height, reader::width, reader::max\_code);

try {

reader::in.close();

} catch (...) {

throw std::runtime\_error("Couldn't close an input file");

}

switch (argv[3][0]) {

case '0':

edit::inverse();

break;

case '1':

edit::mirror::horisontal();

break;

case '2':

edit::mirror::vertical();

break;

case '3':

edit::rotate::clockwise();

break;

case '4':

edit::rotate::counterclockwise();

break;

default:

throw std::runtime\_error("Expected only 0, 1, 2, 3, 4 in command");

break;

}

writer::out.open(argv[2], std::ios::out | std::ios::binary);

if (!writer::out.is\_open()) {

throw std::runtime\_error("An output file can't be opened");

}

writer::write\_meta();

writer::write\_data();

try {

writer::out.close();

} catch (...) {

throw std::runtime\_error("Couldn't close an output file");

}

if (debug) {

using namespace writer;

std::string s = argv[2];

std::ofstream fout(s + "decoded\_from\_output.pnm");

fout << reader::p << reader::type - 3 << '\n' << width << ' ' << height << '\n' << reader::max\_code << '\n';

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

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

for (auto &k : data[i][j].info) {

fout << k << ' ';

}

}

}

fout.close();

}

}

int main(int argc, char\* argv[]) {

try {

run(argc, argv, false);

} catch (std::runtime\_error const& e) {

printf("An error occured\n");

printf(e.what());

printf("\n");

} catch (...) {

printf("An unexpected error occurred\n");

}

delete[] reader::buffer;

return 0;

}