МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ  
 ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНО УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Санкт-Петербургский национальный исследовательский университет  
информационных технологий, механики и оптики»

Факультет информационных технологий и программирования

Компьютерная графика

Лабораторная работа № 1

**Конвертация изображений**

Выполнил: студент группы № М32342  
Кубанцев Ярослав Максимович

Проверил: ассистент факультета прикладной оптики Скаков Павел Сергеевич

САНКТ -ПЕТЕРБУРГ  
2020

**Содержание**

|  |  |
| --- | --- |
| Цель работы | 3 |
| Теоретическая справка | 4 |
| Решение | 6 |
| Листинг программ | 7 |
|  |  |

**Цель работы**

Написать программу, выполняющую простые преобразования изображений в формате PNM.

Программа должна поддерживать серые и цветные изображения (варианты PNM P5 и P6),

самостоятельно определяя формат по содержимому.

**Теоретическая справка**

Растровое изображение — изображение, представляющее собой сетку (мозаику) пикселей — цветных точек (обычно прямоугольных) на мониторе, бумаге и других отображающих устройствах.

Важными характеристиками изображения являются:

1. Размер изображения в пикселях — может выражаться в виде количества пикселей по ширине и по высоте (800 × 600 px, 1024 × 768 px, 1600 × 1200 px и т. д.) или же в виде общего количества пикселей (так, изображение размером 1600 × 1200 px состоит из 1 920 000 точек, то есть примерно из двух мегапикселей);
2. Количество используемых цветов или глубина цвета (эти характеристики имеют следующую зависимость: N=2k, где N — количество цветов, {k — глубина цвета);
3. Цветовое пространство (цветовая модель) — RGB, CMYK, XYZ, YCbCr и др.;
4. Разрешение изображения — величина, определяющая количество точек (элементов растрового изображения) на единицу площади (или единицу длины). Не путать с размером сетки изображения!

Растровую графику редактируют с помощью растровых графических редакторов. Создаётся растровая графика фотоаппаратами, сканерами, непосредственно в растровом редакторе, а также путём экспорта из векторного редактора или в виде снимков экрана.

*Преимущества:*

1. Растровая графика позволяет создать практически любой рисунок, вне зависимости от сложности, в отличие, например, от векторной, где невозможно точно передать эффект перехода от одного цвета к другому без потерь в размере файла;
2. Распространённость — растровая графика используется сейчас практически везде: от маленьких значков до плакатов;
3. Высокая скорость обработки сложных изображений, если не нужно масштабирование;
4. Растровое представление изображения естественно для большинства устройств ввода-вывода графической информации, таких как мониторы (за исключением векторных устройств вывода), матричные и струйные принтеры, цифровые фотоаппараты, сканеры, а также сотовые телефоны.

*Недостатки*

1. Большой размер файлов у простых изображений из большого количества точек;
2. Невозможность идеального масштабирования;
3. Невозможность вывода на печать на векторный графопостроитель.

Простые форматы хранения изображений portable pixmap (иногда определяемые как PNM): цветных (PPM), полутоновых (PGM) и чёрно-белых (PBM) определяют правила для обмена графическими файлами. Эти форматы могут обеспечивать промежуточное представление данных при конвертации растровых графических файлов трёх перечисленных типов между разными платформами.

Формат файла PNM

<Магический номер>

<Ширина> <Высота>

<Максимальное значение>

<Значение пикселей>

Где магическое число – PN, N = 1..6 – обозначение формата данных.

P1 – черно-белое изображение, значение пикселей в виде ASCII цифр.

P2 – полутоное изображение, значение пикселей в виде ASCII цифр.

P3 – цветное изображение RGB, значение пикселей в виде ASCII цифр.

P4 – черно-белое изображение, значение пикселей записаны в отдельные байты.

P5 – полутоное изображение, значение пикселей записаны в отдельные байты.

P6 – цветное изображение RGB, значение пикселей записаны в отдельные байты.

**Решение**

1. Инверсия

В полутоном случае значение каждого пикселя p заменяется на 255 - p. В цветном - такая же операция применяется отдельно для красного, зеленого и синего.

2. Зеркальное отображение по горизонтали

Координаты каждого пикселя (x, y) заменяются на (weight – x - 1, y).

3. Зеркальное отражение по вертикали

Координаты каждого пикселя (x, y) заменяются на (x, height – y - 1).

4. Поворот на 90 градусов по часовой стрелке

Координаты каждого пикселя (x, y) заменяются на (height – y - 1, x).

5. Поворот на 90 градусов против часовой стрелки

Координаты каждого пикселя (x, y) заменяются на (y, weight – x - 1).

**Листинг программ**

*main.cpp*

#include <iostream>  
#include <vector>  
#include <fstream>  
#include <string>  
  
  
struct colored {  
 unsigned char red;  
 unsigned char green;  
 unsigned char blue;  
  
 explicit colored(unsigned char red = 0, unsigned char green = 0, unsigned char blue = 0) : red(red), green(green),  
 blue(blue) {}  
  
};  
  
typedef unsigned char monochrome;  
  
template<typename T>  
std::vector<T> read(std::ifstream &in, size\_t size);  
  
template<>  
std::vector<colored> read(std::ifstream &in, size\_t size) {  
 /\*auto \*buff = new char[size \* 3];  
 in.read(buff, size \* 3);  
 std::vector<colored> data(size);  
 for (size\_t i = 0; i < size; i++) {  
 data[i].red = buff[i \* 3 + 0];  
 data[i].green = buff[i \* 3 + 1];  
 data[i].blue = buff[i \* 3 + 2];  
 }  
 delete[] buff;  
 return data;\*/  
 std::vector<colored> data(size);  
 in.read((char \*) data.data(), size \* 3);  
 return data;  
}  
  
template<>  
std::vector<monochrome> read(std::ifstream &in, size\_t size) {  
 /\*auto \*buff = new char[size];  
 in.read(buff, size);  
 std::vector<monochrome> data(size);  
 for (size\_t i = 0; i < size; i++) {  
 data[i] = buff[i];  
 }  
 delete[] buff;  
 return data;\*/  
 std::vector<monochrome> data(size);  
 in.read((char \*) data.data(), size);  
 return data;  
}  
  
template<typename T>  
void write(std::ofstream &out, std::vector<T> const &data);  
  
template<>  
void write(std::ofstream &out, std::vector<colored> const &data) {  
 out << 255 << std::endl;  
 out.write((char \*) data.data(), data.size() \* 3);  
}  
  
template<>  
void write(std::ofstream &out, std::vector<monochrome> const &data) {  
 out << 255 << std::endl;  
 out.write((char \*) data.data(), data.size());  
}  
  
template<typename T>  
T negate(T pixel);  
  
template<>  
colored negate(colored pixel) {  
 return colored(255 - pixel.red, 255 - pixel.green, 255 - pixel.blue);  
}  
  
template<>  
monochrome negate(monochrome pixel) {  
 return 255 - pixel;  
}  
  
template<typename T>  
void inversionConverse(std::vector<T> &data, size\_t &weight, size\_t &height) {  
 for (size\_t i = 0; i < weight \* height; i++) {  
 data[i] = negate<T>(data[i]);  
 }  
}  
  
template<typename T>  
void verticalConverse(std::vector<T> &data, size\_t &weight, size\_t &height) {  
 for (size\_t i = 0; i < weight; i++) {  
 for (size\_t j = 0; j < height / 2; j++) {  
 std::swap(data[j \* weight + i], data[(height - j - 1) \* weight + i]);  
 }  
 }  
}  
  
template<typename T>  
void horizontalConverse(std::vector<T> &data, size\_t &weight, size\_t &height) {  
 for (size\_t i = 0; i < weight / 2; i++) {  
 for (size\_t j = 0; j < height; j++) {  
 std::swap(data[j \* weight + i], data[j \* weight + (weight - i - 1)]);  
 }  
 }  
}  
  
template<typename T>  
void clockwiseConverse(std::vector<T> &data, size\_t &weight, size\_t &height) {  
 auto newData = std::vector<T>(weight \* height);  
 for (size\_t i = 0; i < weight; i++) {  
 for (size\_t j = 0; j < height; j++) {  
 newData[i \* height + (height - j - 1)] = data[j \* weight + i];  
 }  
 }  
 std::swap(data, newData);  
 std::swap(weight, height);  
}  
  
template<typename T>  
void counterClockwiseConverse(std::vector<T> &data, size\_t &weight, size\_t &height) {  
 auto newData = std::vector<T>(weight \* height);  
 for (size\_t i = 0; i < weight; i++) {  
 for (size\_t j = 0; j < height; j++) {  
 newData[(weight - i - 1) \* height + j] = data[j \* weight + i];  
 }  
 }  
 std::swap(data, newData);  
 std::swap(weight, height);  
}  
  
template<typename T>  
bool converse(std::ifstream &in, std::ofstream &out, size\_t weight, size\_t height, std::string const &conversion) {  
 auto data = read<T>(in, weight \* height);  
 if (conversion == "0") {  
 inversionConverse(data, weight, height);  
 } else if (conversion == "1") {  
 horizontalConverse(data, weight, height);  
 } else if (conversion == "2") {  
 verticalConverse(data, weight, height);  
 } else if (conversion == "3") {  
 clockwiseConverse(data, weight, height);  
 } else if (conversion == "4") {  
 counterClockwiseConverse(data, weight, height);  
 } else {  
 return false;  
 }  
 out << weight << " " << height << std::endl;  
 write<T>(out, data);  
 return true;  
}  
  
int main(int argc, char \*\*argv) {  
 if (argc != 4) {  
 std::cerr << "Use 'converter.exe <input file> <output file> <conversion>'" << std::endl;  
 return 1;  
 }  
 std::ifstream in(argv[1], std::ifstream::binary);  
 std::ofstream out(argv[2], std::ifstream::binary);  
 if (!in.is\_open()) {  
 std::cerr << "Can't open input file";  
 in.close();  
 out.close();  
 return 1;  
 }  
 if (!out.is\_open()) {  
 std::cerr << "Can't open output file";  
 in.close();  
 out.close();  
 return 1;  
 }  
 std::string format;  
 size\_t weight, height, maxValue;  
 if (!(in >> format >> weight >> height >> maxValue)) {  
 std::cerr << "Can't read from input file";  
 in.close();  
 out.close();  
 return 1;  
 }  
 in.get();  
 out << format << std::endl;  
 bool result;  
 try {  
 if (format == "P5" && maxValue == 255) {  
 result = converse<monochrome>(in, out, weight, height, argv[3]);  
 } else if (format == "P6" && maxValue == 255) {  
 result = converse<colored>(in, out, weight, height, argv[3]);  
 } else {  
 std::cerr << "Unknown format of input file" << std::endl;  
 in.close();  
 out.close();  
 return 1;  
 }  
 } catch (std::bad\_alloc &e) {  
 std::cerr << "Not enough memory" << std::endl;  
 in.close();  
 out.close();  
 return 1;  
 }  
 if (!result) {  
 std::cerr << "Unknown conversion" << std::endl;  
 in.close();  
 out.close();  
 return 1;  
 }  
 in.close();  
 out.close();  
 return 0;  
}