**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

Курсовая РАБОТА

**по дисциплине «Программирование»**

Тема: Обработка BMP файла

|  |  |  |
| --- | --- | --- |
| Студент гр. 3341 |  | Кудин А.А. |
| Преподаватель |  | Глазунов С.А. |

Санкт-Петербург

2024

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Кудин А.А. | | |
| Группа 3341  Вариант 12 | | |
| Программа **обязательно должна иметь CLI** (опционально дополнительное использование GUI). Более подробно тут: **http://se.moevm.info/doku.php/courses:programming:rules\_extra\_kurs**  Программа должна реализовывать весь следующий функционал по обработке bmp-файла  **Общие сведения**   * 24 бита на цвет * без сжатия * файл может не соответствовать формату BMP, т.е. необходимо проверка на BMP формат (дополнительно стоит помнить, что версий у формата несколько). Если файл не соответствует формату BMP или его версии, то программа должна завершиться с соответствующей ошибкой. * обратите внимание на выравнивание; мусорные данные, если их необходимо дописать в файл для выравнивания, должны быть нулями. * обратите внимание на порядок записи пикселей * все поля стандартных BMP заголовков в выходном файле должны иметь те же значения что и во входном (разумеется кроме тех, которые должны быть изменены).   Программа должна иметь следующую функции по обработке изображений:     * (1) Рисование квадрата с диагоналями. Флаг для выполнения данной операции: `--squared\_lines`. Квадрат определяется:   + Координатами левого верхнего угла. Флаг `--left\_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y   + Размером стороны. Флаг `--side\_size`. На вход принимает число больше 0   + Толщиной линий. Флаг `--thickness`. На вход принимает число больше 0   + Цветом линий. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)   + Может быть залит или нет (диагонали располагаются “поверх” заливки). Флаг `--fill`. Работает как бинарное значение: флага нет – false , флаг есть – true.   + Цветом которым он залит, если пользователем выбран залитый. Флаг `--fill\_color` (работает аналогично флагу `--color`) * (2) Фильтр rgb-компонент. Флаг для выполнения данной операции: `--rgbfilter`. Этот инструмент должен позволять для всего изображения либо установить в диапазоне от 0 до 255 значение заданной компоненты. Функционал определяется   + Какую компоненту требуется изменить. Флаг `--component\_name`. Возможные значения `red`, `green` и `blue`.   + В какой значение ее требуется изменить. Флаг `--component\_value`. Принимает значение в виде числа от 0 до 255 * (3) Поворот изображения (части) на 90/180/270 градусов. Флаг для выполнения данной операции: `--rotate`. Функционал определяется   + Координатами левого верхнего угла области. Флаг `--left\_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y   + Координатами правого нижнего угла области.  Флаг `--right\_down`, значение задаётся в формате `right.down`, где right – координата по x, down – координата по y   + Углом поворота. Флаг `--angle`, возможные значения: `90`, `180`, `270`   Все подзадачи, ввод/вывод должны быть реализованы в виде отдельной функции. | | |
| Содержание пояснительной записки:  разделы «Аннотация», «Содержание», «Введение», «Ход работы», «Пример работы программы», «Заключение», «Список использованных источников» | | |
| Предполагаемый объем пояснительной записки:  Не менее 15 страниц. | | |
| Дата выдачи задания: 18.03.2024 | | |
| Дата сдачи реферата: 18.05.2024 | | |
| Дата защиты реферата: 23.05.2024 | | |
| Студент |  | Кудин А.А. |
| Преподаватель |  | Глазунов С.А. |

**Аннотация**

В рамках данной курсовой работы была создана программа для обработки изображений в формате BMP. Программа проверяет формат и параметры изображения, а при соответствии заданным условиям, выполняет необходимые операции и выводит изменённую копию изображения. Взаимодействие с программой осуществляется через командную строку (CLI).

**содержание**

|  |  |  |
| --- | --- | --- |
|  | Введение | 7 |
| 1. | Работа с файлами | 8 |
| 2. | Ввод аргументов | 9 |
| 3. | Обработка изображения | 10 |
|  | Заключение | 14 |
|  | Список использованных источников | 15 |
|  | Приложение А. Исходный код программы | 16 |
|  | Приложение Б. Тестирование | 35 |
|  |  |  |

**введение**

Цель работы заключается в создании программы для обработки BMP-файлов с использованием командной строки (CLI) и, при необходимости, графического интерфейса пользователя (GUI). Программа должна проверять соответствие файла формату BMP, учитывая различные версии, и выполнять следующие функции:

Проверка файла:

Убедиться, что файл является BMP.

Проверить версию BMP-файла.

Обеспечить корректное выравнивание данных в файле.

Сохранить значения всех полей стандартных заголовков BMP.

Обработка изображений:

Рисование квадрата с диагоналями (--squared\_lines):

Координаты левого верхнего угла (--left\_up).

Размер стороны (--side\_size).

Толщина линий (--thickness).

Цвет линий (--color).

Заливка (--fill).

Цвет заливки (--fill\_color).

Фильтр RGB-компонент (--rgbfilter):

Компонента для изменения (--component\_name).

Значение компоненты (--component\_value).

Поворот изображения (--rotate):

Координаты левого верхнего угла (--left\_up).

Координаты правого нижнего угла (--right\_down).

Угол поворота (--angle).

**1. Работа с файлами**

Чтение BMP-файлов:

Функция isBMP проверяет, является ли файл формата BMP, читая его сигнатуру.

Функция readBmp загружает BMP-файл, читая заголовки и пиксельные данные, и сохраняет их в соответствующих структурах.

Запись BMP-файлов:

Функция writeBmp записывает измененные данные изображения в новый BMP-файл, сохраняя при этом все поля стандартных заголовков BMP.

**2. ВВОД АРГУМЕНТОВ**

В данной программе реализована обработка аргументов командной строки с использованием CLI (Command Line Interface). Для обработки аргументов командной строки используются структуры option, которые определяют различные действия, доступные в программе.

Для каждой основной команды (например, squared\_lines, rgbfilter, rotate) определены соответствующие наборы опций командной строки. Например, для команды squared\_lines опции определены в структуре squared\_linesKeys, для команды rgbfilter — в структуре rgbFilterKeys, а для команды rotate — в структуре rotateKeys.

Функция findUnknownKey осуществляет анализ аргументов командной строки и проверяет наличие неизвестных ключей. В случае обнаружения неизвестного ключа программа выводит сообщение об ошибке и завершает работу.

Функция processCommand выполняет разбор аргументов командной строки и выбор нужного действия в зависимости от команды. Она вызывает соответствующую функцию обработки в зависимости от команды, такие как processSquareWithDiagonalsCommand, processRGBFilter и processRotate.

Каждая функция обработки команды (например, processSquareWithDiagonalsCommand, processRGBFilter, processRotate) осуществляет разбор опций командной строки и вызывает соответствующую функцию для выполнения задачи. В случае неверных данных или ошибочных аргументов функции выводят сообщение об ошибке и завершают работу программы с соответствующим кодом ошибки.

Таким образом, пользователь может использовать CLI для выполнения различных действий с BMP-файлами, таких как рисование квадратов с диагоналями, применение RGB-фильтров или поворот изображения, передавая соответствующие аргументы командной строки.

**3. ОСНОВНЫЕ ФУНКЦИИ**

**3.1 Функция checkValidCoord**

Проверяет, являются ли указанные координаты действительными для текущего изображения, проверяя, находятся ли координаты в пределах допустимых значений для ширины и высоты изображения.

**3.2 Рисование фигур**

Функция drawCircle рисует круг заданного цвета и толщины на изображении. Функция drawLine рисует линию между двумя точками с заданной толщиной и цветом, используя алгоритм Брезенхэма. Функция drawRect рисует прямоугольник с заданной толщиной и цветом, а при необходимости также заполняет его указанным цветом. Функция drawSquareWithDiagonals рисует квадрат с диагоналями, определяя координаты вершин квадрата, рисуя его стороны и проводя диагональные линии.

**3.3 Заполнение прямоугольника**

Функция fillRect заполняет прямоугольную область заданным цветом, устанавливая цвет каждого пикселя в пределах указанного прямоугольника.

**3.4 Применение RGB-фильтра**

Функция applyRGBFilter изменяет значение заданной RGB-компоненты (красной, зеленой или синей) для каждого пикселя на изображении в соответствии с указанным значением.

**3.5 Поворот изображения**

Функция rotate выполняет поворот выделенной области изображения на заданный угол (90, 180 или 270 градусов), вычисляя новые координаты пикселей после поворота и сохраняя измененные данные в буфер, который затем заменяет исходный массив пикселей.

**3. 6 Преобразование координат и проверка имени файла**

Функция convertCoords преобразует строковые координаты в целочисленные значения, проверяя формат строки и извлекая значения координат x и y. Функция checkValidName проверяет, соответствует ли имя файла допустимому формату, используя регулярное выражение.

**3.7 Вспомогательные функции**

Функция description выводит описание программы, указывая, что это курсовая работа для опции 4.12, созданная Кудиным Александром. Функция outputHelp выводит справочную информацию по использованию программы, описывая доступные команды и параметры. Функция outputInfo выводит информацию о BMP-файле, включая разрешение, размер файла, глубину цвета и количество цветовых плоскостей.

**3.8 Обработка команд**

Функция processSquareWithDiagonalsCommand обрабатывает команду для рисования квадрата с диагоналями, считывая параметры командной строки и вызывая соответствующие функции для выполнения операции. Функция processRGBFilter обрабатывает команду для применения RGB-фильтра, считывая параметры командной строки и применяя фильтр к изображению. Функция processRotate обрабатывает команду для поворота изображения, считывая параметры командной строки и выполняя поворот указанной области изображения. Функция findUnknownKey проверяет наличие неизвестных ключей в командной строке, выводя сообщение об ошибке и завершая работу при обнаружении неизвестного ключа. Функция processCommand обрабатывает основную команду, переданную программе через командную строку, вызывая соответствующие функции для выполнения команд.

**заключение**

В ходе выполнения данного проекта была разработана программа для обработки изображений в формате BMP. Программа имеет командную строку (CLI), что обеспечивает удобство взаимодействия с пользователем. Она реализует следующий функционал:

* **Рисование квадрата с диагоналями**: Пользователь может указать координаты левого верхнего угла квадрата, размер стороны, толщину линий, цвет линий, а также цвет заливки, если квадрат залит.
* **Фильтр RGB-компонент**: Программа позволяет изменять значения RGB-компонент (красный, зеленый, синий) для всего изображения, задавая нужные значения в диапазоне от 0 до 255.
* **Поворот изображения**: Пользователь может повернуть выделенную область изображения на 90, 180 или 270 градусов, указав координаты левого верхнего и правого нижнего углов области.

Важным аспектом является обработка входных данных и валидация параметров пользовательского ввода. Программа проверяет соответствие входного изображения формату BMP, корректность всех переданных параметров и выравнивание данных в файле.

Все подзадачи, такие как рисование квадрата, применение RGB-фильтра и поворот изображения, реализованы в виде отдельных функций, что способствует модульности и повторному использованию кода. Программа сохраняет все поля стандартных BMP-заголовков с соответствующими значениями и корректно обрабатывает мусорные данные для выравнивания.

**список использованных источников**

1.https://se.moevm.info/lib/exe/fetch.php/courses:programming:programming\_cw\_metoda\_2nd\_course\_last\_ver.pdf.pdf - методические материалы для написания курсовой работы

**приложение А**

**Исходный код программы**

Название файла: main.cpp

#include <iostream>

#include <fstream>

#include <cmath>

#include <cstring>

#include <getopt.h>

#include <regex>

#include <vector>

#pragma pack(push, 1)

typedef struct {

unsigned short signature;

unsigned int filesize;

unsigned short reserved1;

unsigned short reserved2;

unsigned int pixelArrOffset;

} BitmapFileHeader;

typedef struct {

unsigned int headerSize;

unsigned int width;

unsigned int height;

unsigned short planes;

unsigned short bitsPerPixel;

unsigned int compression;

unsigned int imageSize;

unsigned int xPixelsPerMeter;

unsigned int yPixelsPerMeter;

unsigned int colorsInColorTable;

unsigned int importantColorCount;

} BitmapInfoHeader;

typedef struct {

unsigned char b;

unsigned char g;

unsigned char r;

} Rgb;

#pragma pack(pop)

static struct option keys[] = {

{"help", no\_argument, 0, 'h'},

{"output", required\_argument, 0, 'o'},

{"input", required\_argument, 0, 'i'},

{"info", no\_argument, 0, 'd'},

{"squared\_lines", no\_argument, 0, 's'},

{"rgbfilter", no\_argument, 0, 'e'},

{"rotate", no\_argument, 0, 'r'},

{"left\_up", required\_argument, 0, 'u'},

{"right\_down", required\_argument, 0, 'q'},

{"side\_size", required\_argument, 0, 'z'},

{"thickness", required\_argument, 0, 't'},

{"color", required\_argument, 0, 'c'},

{"fill", no\_argument, 0, 'f'},

{"fill\_color", required\_argument, 0, 'g'},

{"component\_name", required\_argument, 0, 'n'},

{"component\_value", required\_argument, 0, 'v'},

{"angle", required\_argument, 0, 'a'},

{0, 0, 0, 0}

};

static struct option selectAction[] = {

{"help", no\_argument, 0, 'h'},

{"output", required\_argument, 0, 'o'},

{"input", required\_argument, 0, 'i'},

{"info", no\_argument, 0, 'd'},

{"squared\_lines", no\_argument, 0, 's'},

{"rgbfilter", no\_argument, 0, 'e'},

{"rotate", no\_argument, 0, 'r'},

{0, 0, 0, 0}

};

static struct option squared\_linesKeys[] = {

{"left\_up", required\_argument, 0, 'u'},

{"side\_size", required\_argument, 0, 's'},

{"thickness", required\_argument, 0, 't'},

{"color", required\_argument, 0, 'c'},

{"fill", no\_argument, 0, 'f'},

{"fill\_color", required\_argument, 0, 'g'},

{0, 0, 0, 0}

};

static struct option rgbFilterKeys[] = {

{"component\_name", required\_argument, 0, 'n'},

{"component\_value", required\_argument, 0, 'v'},

{0, 0, 0, 0}

};

static struct option rotateKeys[] = {

{"left\_up", required\_argument, 0, 'u'},

{"right\_down", required\_argument, 0, 'd'},

{"angle", required\_argument, 0, 'a'},

{0, 0, 0, 0}

};

bool isBMP(const std::string& filename) {

std::ifstream file(filename, std::ios::binary);

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

return false;

}

char header[2];

file.read(header, 2);

bool isBmp = (header[0] == 'B' && header[1] == 'M');

file.close();

return isBmp;

}

bool readBmp(std::string fileName, BitmapFileHeader &header,

BitmapInfoHeader &infoHeader, Rgb\*\* &pixelArray){

std::fstream file(fileName, std::ios::binary | std::ios::in);

try {

if (!file.is\_open()) return false;

file.read(reinterpret\_cast<char\*>(&header), sizeof(BitmapFileHeader));

file.read(reinterpret\_cast<char\*>(&infoHeader), sizeof(BitmapInfoHeader));

int rowSize = ((infoHeader.width \* sizeof(Rgb)) + 3) & (~3);

pixelArray = new Rgb\*[infoHeader.height];

for (int i = infoHeader.height - 1; i >= 0; i--) {

pixelArray[i] = new Rgb[infoHeader.width];

file.read(reinterpret\_cast<char\*>(pixelArray[i]), rowSize);

}

file.close();

} catch (const std::bad\_alloc& a) {

return false;

}

if (!isBMP(fileName)) return false;

return true;

}

void writeBmp(std::string fileName, BitmapFileHeader &header,

BitmapInfoHeader &infoHeader, Rgb\*\* &pixelArray){

std::fstream file(fileName, std::ios::binary | std::ios::out);

file.write(reinterpret\_cast<char\*>(&header), sizeof(BitmapFileHeader));

file.write(reinterpret\_cast<char\*>(&infoHeader), sizeof(BitmapInfoHeader));

int rowSize = ((infoHeader.width \* sizeof(Rgb)) + 3) & (~3);

for (int i = infoHeader.height - 1; i >= 0; i--) {

file.write(reinterpret\_cast<char\*>(pixelArray[i]), rowSize);

}

file.close();

}

bool checkValidCoord(int& x, int& y, BitmapInfoHeader &infoHeader){

if (static\_cast<unsigned int>(x) >= infoHeader.width || x < 0) return false;

if (static\_cast<unsigned int>(y) >= infoHeader.height || y < 0) return false;

return true;

}

void getRgb(std::string &color, int\* arrayRgb){

char colorRgb[color.size()+1];

char\* pointer;

strcpy(colorRgb, color.c\_str());

pointer = strtok(colorRgb, ".");

arrayRgb[0] = atoi(pointer);

pointer = strtok(NULL, ".");

arrayRgb[1] = atoi(pointer);

pointer = strtok(NULL, ".");

arrayRgb[2] = atoi(pointer);

}

void drawCircle(int& x, int& y, int thickness, std::string& color,

Rgb\*\* &pixelArray, BitmapInfoHeader &infoHeader){

int r = thickness / 2;

int arrayRgb[3];

getRgb(color, arrayRgb);

if (r < 1) {

if (checkValidCoord(x, y, infoHeader)) {

pixelArray[y][x].r = arrayRgb[0];

pixelArray[y][x].g = arrayRgb[1];

pixelArray[y][x].b = arrayRgb[2];

}

return;

}

int xc = r;

int yc = 0;

int P = 1 - r;

while (xc >= yc) {

int xMinusXC = x - xc;

int xPlusXC = x + xc;

int yPlusYC = y + yc;

int yMinusYC = y - yc;

int yPlusXC = y + xc;

int xPlusYC = x + yc;

int xMinusYC = x - yc;

int yMinusXC = y - xc;

for (int i = xMinusXC; (i <= xPlusXC) && (i < static\_cast<int>(infoHeader.width)); i++) {

if ((yPlusYC >= 0) && (i >= 0) && (yPlusYC < static\_cast<int>(infoHeader.height))) {

pixelArray[yPlusYC][i].r = arrayRgb[0];

pixelArray[yPlusYC][i].g = arrayRgb[1];

pixelArray[yPlusYC][i].b = arrayRgb[2];

}

if ((yMinusYC >= 0) && (i >= 0) && (yMinusYC < static\_cast<int>(infoHeader.height))) {

pixelArray[yMinusYC][i].r = arrayRgb[0];

pixelArray[yMinusYC][i].g = arrayRgb[1];

pixelArray[yMinusYC][i].b = arrayRgb[2];

}

}

for (int i = xMinusYC; (i <= xPlusYC) && (i < static\_cast<int>(infoHeader.width)); i++) {

if ((yPlusXC >= 0) && (i >= 0) && (yPlusXC < static\_cast<int>(infoHeader.height))) {

pixelArray[yPlusXC][i].r = arrayRgb[0];

pixelArray[yPlusXC][i].g = arrayRgb[1];

pixelArray[yPlusXC][i].b = arrayRgb[2];

}

if ((yMinusXC >= 0) && (i >= 0) && (yMinusXC < static\_cast<int>(infoHeader.height))) {

pixelArray[yMinusXC][i].r = arrayRgb[0];

pixelArray[yMinusXC][i].g = arrayRgb[1];

pixelArray[yMinusXC][i].b = arrayRgb[2];

}

}

yc++;

if (P <= 0) {

P = P + 2 \* yc + 1;

} else {

xc--;

P = P + 2 \* (yc - xc) + 1;

}

}

}

void drawLine(int firstX, int firstY, int secondX, int secondY, int thickness,

std::string& color, Rgb\*\* &pixelArray, BitmapInfoHeader &infoHeader){

int dx = abs(secondX - firstX);

int dy = abs(secondY - firstY);

int sx = firstX < secondX ? 1 : -1;

int sy = firstY < secondY ? 1 : -1;

int err = dx - dy;

int x = firstX;

int y = firstY;

while (x != secondX || y != secondY) {

drawCircle(x, y, thickness, color, pixelArray, infoHeader);

int err2 = 2 \* err;

if (err2 > -dy) {

err -= dy;

x += sx;

}

if (err2 < dx) {

err += dx;

y += sy;

}

}

drawCircle(secondX, secondY, thickness, color, pixelArray, infoHeader);

}

void fillRect(int leftX, int leftY, int rightX, int rightY, int thickness, Rgb\*\* &pixelArray, std::string &fillColor, BitmapInfoHeader &infoHeader){

int arrayFillRgb[3];

getRgb(fillColor, arrayFillRgb);

for (int i = leftX; i <= rightX; i++) {

for (int k = leftY; k <= rightY; k++) {

if (checkValidCoord(i, k, infoHeader)) {

pixelArray[k][i].r = arrayFillRgb[0];

pixelArray[k][i].g = arrayFillRgb[1];

pixelArray[k][i].b = arrayFillRgb[2];

}

}

}

}

void drawRect(int leftX, int leftY, int rightX, int rightY, int thickness,

std::string& color, bool fill, std::string fillColor, Rgb\*\* &pixelArray, BitmapInfoHeader &infoHeader){

if (fill == true) fillRect(leftX, leftY, rightX, rightY, thickness, pixelArray, fillColor, infoHeader);

drawLine(leftX, leftY, rightX, leftY, thickness, color, pixelArray, infoHeader);

drawLine(rightX, leftY, rightX, rightY, thickness, color, pixelArray, infoHeader);

drawLine(leftX, rightY, rightX, rightY, thickness, color, pixelArray, infoHeader);

drawLine(leftX, leftY, leftX, rightY, thickness, color, pixelArray, infoHeader);

}

void drawSquareWithDiagonals(int leftX, int leftY, int sideSize, int thickness,

std::string color, bool fill, std::string fillColor,

Rgb\*\* &pixelArray, BitmapInfoHeader &infoHeader) {

drawRect(leftX, leftY, leftX + sideSize, leftY + sideSize, thickness, color, fill, fillColor, pixelArray, infoHeader);

drawLine(leftX, leftY, leftX + sideSize, leftY + sideSize, thickness, color, pixelArray, infoHeader);

drawLine(leftX + sideSize, leftY, leftX, leftY + sideSize, thickness, color, pixelArray, infoHeader);

}

void applyRGBFilter(Rgb\*\* &pixelArray, BitmapInfoHeader &infoHeader, const std::string& component\_name, int component\_value) {

int index = (component\_name == "red") ? 2 : (component\_name == "green" ? 1 : 0);

for (unsigned int y = 0; y < infoHeader.height; ++y) {

for (unsigned int x = 0; x < infoHeader.width; ++x) {

unsigned char\* component = reinterpret\_cast<unsigned char\*>(&pixelArray[y][x]) + index;

\*component = static\_cast<unsigned char>(component\_value);

}

}

}

void rotate(Rgb\*\* &pixelArray, BitmapInfoHeader &infoHeader, int coords[2][2], int angle) {

using std::min, std::max;

// Координаты выделенной области

int left = coords[0][0];

int top = coords[0][1];

int right = coords[1][0];

int bottom = coords[1][1];

// Размеры выделенной области

int Rwidth = right - left + 1;

int Rheight = bottom - top + 1;

int centerX = left + Rwidth / 2;

int centerY = top + Rheight / 2;

// Создаем временный массив для поворота

Rgb\*\* buffer = new Rgb\*[infoHeader.height];

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

buffer[i] = new Rgb[infoHeader.width];

std::copy(pixelArray[i], pixelArray[i] + infoHeader.width, buffer[i]);

}

// Поворот изображений

for (unsigned int y = 0; y < infoHeader.height; ++y) {

for (unsigned int x = 0; x < infoHeader.width; ++x) {

int oldX = 0, oldY = 0;

int delta\_x = 0, delta\_y = 0;

switch (angle) {

case 270:

oldX = centerX + (y - centerY);

oldY = centerY - (x - centerX);

if ((Rheight % 2) != (Rwidth % 2)) {

if ((Rheight % 2)) {

delta\_y = -1 + (Rheight % 2);

delta\_x = -(Rwidth % 2);

} else {

delta\_y = -(Rheight % 2);

delta\_x = -(Rwidth % 2);

}

} else {

delta\_y = -((1 + Rheight) % 2);

delta\_x = -((1 + Rwidth) % 2);

}

break;

case 180:

oldX = centerX - (x - centerX);

oldY = centerY - (y - centerY);

if (!(Rheight % 2) && !(Rwidth % 2)) {

delta\_x = -1 + (Rheight % 2);

delta\_y = -1 + (Rwidth % 2);

} else {

delta\_x = -(Rheight % 2);

delta\_y = -(Rwidth % 2);

}

break;

case 90:

oldX = centerX - (y - centerY);

oldY = centerY + (x - centerX);

delta\_y = -1 + (Rwidth % 2);

delta\_x = -(Rheight % 2);

break;

default:

return;

}

// Проверка, находятся ли старые координаты в пределах буфера

if ((oldX < 0 || oldY < 0 || oldX >= static\_cast<int>(infoHeader.width) || oldY >= static\_cast<int>(infoHeader.height)) ||

(oldY < top || oldY >= bottom || oldX < left || oldX >= right))

continue;

Rgb old\_pix = pixelArray[oldY][oldX];

if (y + delta\_y >= 0 && static\_cast<unsigned int>(y + delta\_y) < infoHeader.height && x + delta\_x >= 0 && static\_cast<unsigned int>(x + delta\_x) < infoHeader.width) {

buffer[y + delta\_y][x + delta\_x] = old\_pix;

}

}

}

// Замена старого пиксельного массива новым

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

delete[] pixelArray[i];

}

delete[] pixelArray;

pixelArray = buffer;

}

bool convertCoords(std::string stringCoords, int &x, int &y){

if (!std::regex\_match(stringCoords.c\_str(), std::regex("(-?[0-9]+).(-?[0-9]+)"))) return false;

x = atoi(stringCoords.c\_str());

int i = 0;

while (true) {

if (stringCoords[i] == '.') break;

i++;

}

y = atoi(i + 1 + stringCoords.c\_str());

return true;

}

bool checkValidName(std::string name){

return (std::regex\_match(name, std::regex("((./)||((\\w+/)+))?(\\w+).(\\w+)")));

}

void description(){

std::cout << "Course work for option 4.12, created by Kudin Aleksandr." << std::endl;

}

void outputHelp() {

std::cout << "Help: данная программа предназначена для обработки BMP изображения." << std::endl <<

"Использование:" << std::endl <<

"--output [filename] : задать имя выходного файла (по умолчанию out.bmp)" << std::endl <<

"--input [filename] : задать имя входного BMP файла" << std::endl <<

"--info : вывод информации о BMP файле" << std::endl <<

"--squared\_lines : нарисовать квадрат с диагоналями" << std::endl <<

"--rgbfilter : применить RGB фильтр к изображению" << std::endl <<

"--rotate : повернуть изображение" << std::endl <<

"Примеры ключей:" << std::endl <<

"--left\_up x,y : координаты левого верхнего угла" << std::endl <<

"--right\_down x,y : координаты правого нижнего угла" << std::endl <<

"--side\_size [size] : размер стороны квадрата" << std::endl <<

"--thickness [size] : толщина линий" << std::endl <<

"--color rrr,ggg,bbb : цвет в формате RGB" << std::endl <<

"--fill : указывает, что фигура должна быть заполнена" << std::endl <<

"--fill\_color rrr,ggg,bbb : цвет заливки фигуры" << std::endl <<

"--angle [degrees] : угол поворота изображения" << std::endl;

}

void outputInfo(BitmapInfoHeader &infoHeader){

printf("BMP file info:\n1) Разрешение файла %dx%d.\n", infoHeader.width, infoHeader.height);

printf("2) Размер файла: %d байтов\n", infoHeader.imageSize);

printf("3) Глубина изображения: %d\n", infoHeader.bitsPerPixel);

printf("4) Количестов цветовых плоскостей: %d\n", infoHeader.planes);

}

void processSquareWithDiagonalsCommand(BitmapInfoHeader &infoHeader, Rgb\*\* &pixelArray, int argc, char\* argv[]) {

int opt, keyIndex;

int leftX = -1, leftY = -1, sideSize = -1, thickness = -1;

std::string color = "", fillColor = "";

bool fill = false;

while ((opt = getopt\_long(argc, argv, "", squared\_linesKeys, &keyIndex)) != -1) {

switch (opt) {

case 'u':

convertCoords(optarg, leftX, leftY);

break;

case 's':

sideSize = std::atoi(optarg);

if (sideSize <= 0) {

std::cerr << "Ошибка: Размер стороны должен быть положительным числом." << std::endl;

exit(41);

}

break;

case 't':

thickness = std::atoi(optarg);

if (thickness <= 0) {

std::cerr << "Ошибка: Толщина должна быть положительным числом." << std::endl;

exit(41);

}

break;

case 'c':

if (std::regex\_match(optarg, std::regex("(25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9]).(25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9]).(25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9])")))

color = optarg;

break;

case 'f':

fill = true;

break;

case 'g':

if (std::regex\_match(optarg, std::regex("(25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9]).(25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9]).(25[0-5]|2[0-4][0-9]|[01]?[0-9]?[0-9])")))

fillColor = optarg;

break;

}

}

if (leftX == -1 || leftY == -1 || sideSize == -1 || thickness == -1 || color.empty()) {

std::cerr << "Ошибка: Не все обязательные параметры были предоставлены." << std::endl;

exit(41);

}

if (fill && fillColor.empty()) {

std::cerr << "Ошибка: Заливка указана, но цвет заливки не предоставлен." << std::endl;

exit(41);

}

int rightX = leftX + sideSize;

int rightY = leftY + sideSize;

if (leftX > rightX) {

std::swap(leftX, rightX);

}

if (leftY > rightY) {

std::swap(leftY, rightY);

}

drawSquareWithDiagonals(leftX, leftY, sideSize, thickness, color, fill, fillColor, pixelArray, infoHeader);

}

void processRGBFilter(BitmapInfoHeader &infoHeader, Rgb\*\* &pixelArray, int argc, char\* argv[]) {

int opt;

int keyIndex;

std::string component\_name = "";

int component\_value = -1;

while (true) {

opt = getopt\_long(argc, argv, "", rgbFilterKeys, &keyIndex);

if (opt == -1) {

break;

}

switch (opt) {

case 'n':

component\_name = optarg;

break;

case 'v':

component\_value = std::stoi(optarg);

if (component\_value < 0 || component\_value > 255) {

std::cout << "Component value must be between 0 and 255" << std::endl;

exit(41);

}

break;

}

}

if (component\_name.empty() || component\_value == -1) {

std::cout << "Invalid or incomplete data for RGB filter" << std::endl;

exit(41);

}

applyRGBFilter(pixelArray, infoHeader, component\_name, component\_value);

}

void processRotate(BitmapInfoHeader &infoHeader, Rgb\*\* &pixelArray, int argc, char\* argv[]) {

int opt;

int keyIndex;

int coords[2][2] = {{-1, -1}, {-1, -1}};

int angle = 0;

while (true) {

opt = getopt\_long(argc, argv, "", rotateKeys, &keyIndex);

if (opt == -1) {

break;

}

switch (opt) {

case 'u':

if (!convertCoords(optarg, coords[0][0], coords[0][1])) {

std::cout << "Invalid format for left\_up coordinates" << std::endl;

exit(41);

}

break;

case 'd':

if (!convertCoords(optarg, coords[1][0], coords[1][1])) {

std::cout << "Invalid format for right\_down coordinates" << std::endl;

exit(41);

}

break;

case 'a':

try {

angle = std::stoi(optarg);

} catch (std::invalid\_argument&) {

std::cout << "Angle must be an integer value" << std::endl;

exit(41);

}

if (angle != 90 && angle != 180 && angle != 270) {

std::cout << "Angle must be 90, 180, or 270 degrees" << std::endl;

exit(41);

}

break;

}

}

if (coords[0][0] == -1 || coords[0][1] == -1 || coords[1][0] == -1 || coords[1][1] == -1) {

std::cout << "Coordinates for both corners must be specified" << std::endl;

exit(41);

}

rotate(pixelArray, infoHeader, coords, angle);

}

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

int keyIndex;

int opt;

char\*\* argvCopy = new char\*[argc];

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

size\_t len = strlen(argv[i]) + 1;

argvCopy[i] = new char[len];

strncpy(argvCopy[i], argv[i], len);

}

while (true) {

opt = getopt\_long(argc, argvCopy, "io:h", keys, &keyIndex);

if (opt == -1) {

opt = 0;

break;

}

switch (opt) {

case '?':

optind = 1;

return true;

break;

}

}

optind = 1;

return false;

}

bool processCommand(BitmapFileHeader& header, BitmapInfoHeader &infoHeader, Rgb\*\* &pixelArray, int argc, char\* argv[], std::string &outputName, std::string &inputName) {

int opt;

int keyIndex;

opterr = 0;

bool validFileName = false;

bool squaredLines = false;

bool rgbFilter = false;

bool rotate = false;

bool printInfo = false;

int count = 0;

char\*\* argvCopy = new char\*[argc];

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

size\_t len = strlen(argv[i]) + 1;

argvCopy[i] = new char[len];

strncpy(argvCopy[i], argv[i], len);

}

if (argc == 1) outputHelp();

while (true) {

opt = getopt\_long(argc, argvCopy, "ho:ireds", selectAction, &keyIndex);

if (opt == -1) {

opt = 0;

break;

}

switch (opt) {

case 'h':

outputHelp();

break;

case 'o':

if (checkValidName(optarg)) outputName = optarg;

break;

case 'i':

if (checkValidName(optarg)) {

if (readBmp(optarg, header, infoHeader, pixelArray)) validFileName = true;

inputName = optarg;

}

break;

case 'd':

printInfo = true;

break;

case 's':

squaredLines = true;

count++;

break;

case 'e':

rgbFilter = true;

count++;

break;

case 'r':

rotate = true;

count++;

break;

}

}

if (printInfo) {

if (validFileName) outputInfo(infoHeader);

else {

std::cout << "Error: the input file name is invalid or the input file is corrupted" << std::endl;

exit(41);

}

}

optind = 1;

if (count > 1) {

std::cout << "Error: too many arguments" << std::endl;

exit(41);

}

else {

if (validFileName) {

if (squaredLines) processSquareWithDiagonalsCommand(infoHeader, pixelArray, argc, argv);

if (rgbFilter) processRGBFilter(infoHeader, pixelArray, argc, argv);

if (rotate) processRotate(infoHeader, pixelArray, argc, argv);

}

else if (squaredLines || rgbFilter || rotate) {

std::cout << "Invalid input file name" << std::endl;

exit(41);

}

}

return validFileName;

}

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

BitmapFileHeader header;

BitmapInfoHeader infoHeader;

Rgb\*\* pixelArray;

std::string outputName = "out.bmp";

std::string inputName = "";

description();

if (findUnknownKey(argc, argv)) {

std::cout << "Error: unknown key provided\n";

exit(41);

}

if (processCommand(header, infoHeader, pixelArray, argc, argv, outputName, inputName)) {

if (outputName == inputName) {

std::cout << "Error: the names of the input and output files are the same\n";

exit(41);

} else {

writeBmp(outputName, header, infoHeader, pixelArray);

}

}

return 0;

}

**приложение б**

**Тестирование**

Рисование квадрата с диагоналями: .\a.exe --squared\_lines --left\_up 20.20 --side\_size 30 --thickness 3 --color 255.0.0 --input fd.bmp

Рисунок 1. Входное изображение



Рисунок 2. Выходное изображение



RGB-фильтр:.\a.exe --rgbfilter --component\_name green --component\_value 0 --input fd.bmp

Рисунок 3. Входное изображение



Рисунок 4. Выходное изображение



Поворот изображения: .\a.exe --rotate --left\_up 98.136 --right\_down 369.255 --angle 180 --input fd.bmp

Рисунок 5. Входное изображение



Рисунок 6. Выходное изображение