**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 |  | Рябов М.Л. |
| Преподаватель |  | Глазунов С.А. |

Санкт-Петербург

2024

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Рябов М.Л. | | |
| 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. Флаг для выполнения данной операции: `--square`. 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. 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) Поменять местами 4 куска области. Флаг для выполнения данной операции: `--exchange`. Выбранная пользователем прямоугольная область делится на 4 части и эти части меняются местами. 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  Способом обмена частей: “по кругу”, по диагонали. Флаг `--exchange\_type`, возможные значения: `clockwise`, `counterclockwise`, `diagonals`  (3) Находит самый часто встречаемый цвет и заменяет его на другой заданный цвет. Флаг для выполнения данной операции: `--freq\_color`. Функционал определяется:  Цветом, в который надо перекрасить самый часто встречаемый цвет. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)  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: 29.05.2024 | | |
| Студент |  | Рябов М.Л. |
| 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 (--square):

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

Размер стороны (--side\_size).

Толщина линий (--thickness).

Цвет линий (--color).

Заливка (--fill).

Цвет заливки (--fill\_color).

Поменять местами 4 куска изображения (--exchange):

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

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

Способ обмена частей (--exchange\_type)

Нахождение самого часто встречаемого цвета (--freq\_color)

Замена самого частого цвета на другой (--color)

**1. Работа с файлами**

Чтение BMP-файлов:

Функция readBMP проверяет, является ли файл формата BMP, читая его сигнатуру и 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, square, freq\_color, exchange) определены соответствующие наборы опций командной строки. Например, для всех команд опции определены в структуре main\_options

Функция getKeys осуществляет анализ аргументов командной строки и проверяет наличие неизвестных ключей. В случае обнаружения неизвестного ключа программа выводит сообщение об ошибке и завершает работу также она выполняет разбор аргументов командной строки и выбор нужного действия в зависимости от команды. Она вызывает соответствующую функцию обработки в зависимости от команды, такие как drawSquare, findAndReplaceFreqColor

Каждая функция обработки команды 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 parseCoords**

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 drawLineByBrezenhem рисует линию между двумя точками с заданной толщиной и цветом, используя алгоритм Брезенхэма. Функция drawSquare рисует квадрат с заданной толщиной и цветом, а при необходимости также заполняет его указанным цветом.

**3.3 Заполнение квадрата**

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

**3.4 Поменять 4 куска области**

Функция exchangeAreas с помощью функции checkCorrectPlacemenе проверяет,все ли координаты соответствуют заданной области,с помощью функции copyArea копирует 4 куска выделенной области, при помощи функции isCorrectType проверяет, корректность звдвнного типа передвижения кусков облати и в зависимости от него использует функцию pasteArea

**3.5 нахождение самого частого цвета**

Функция findAndReplaceColor при помощи функции GetPrimeColor находит самый частый цвет в изображении и возвращает его, после функция снова проходится по всем пикселям и если он совпадает с самым частым, то меняет его на тот, который был передан с помощью флага.

**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

**3.7 Вспомогательные функции**

Функция callInfo выводит описание программы, указывая, что это курсовая работа для опции 4.4, созданная Михаилом Рябовым. Также функция выводит информацию о BMP-файле, включая разрешение, размер файла, глубину цвета и количество цветовых плоскостей. Функция callHelp выводит справочную информацию по использованию программы, описывая доступные команды и параметры.

**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: Пользователь может указать координаты левого верхнего угла квадрата, размер стороны, толщину линий, цвет линий, а также цвет заливки, если квадрат залит.

Фильтр freq\_color: Программа ищет самый частый цвет и изменет значения RGB-компонент (красный, зеленый, синий) для всего изображения, задавая нужные значения в диапазоне от 0 до 255.

Замена 4 частей области: программа может, в зависимости от типа и заданной области поменять местами 4 куска 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.c

#include <stdio.h>

#include <stdlib.h>

#include <regex.h>

#include <string.h>

#include <getopt.h>

#include <math.h>

typedef unsigned long dword;

typedef unsigned short word;

typedef unsigned char BYTE;

#pragma pack (push, 1) //Set padding in 1 byte для того чтобы они лежали подряд для fread

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) //удаление со стека единичного выравнивания

typedef struct Map {

Rgb\* color;

int\* count;

int selfSize;

int capacity;

} Map;

typedef struct SquareProperty

{

unsigned int sumStatusKeys;

int left;

int up;

unsigned int side\_size;

unsigned int thickness;

Rgb color;

Rgb fill\_color;

} SquareProperty;

struct option main\_keys[] = {

{"info", no\_argument, NULL, 'I'},

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

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

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

{"square", no\_argument, NULL, 'S'},

{"left\_up", required\_argument, NULL, 'l'},

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

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

{"color", required\_argument, NULL, 'C'},

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

{"fill\_color", required\_argument, NULL, 'c'},

{"exchange", no\_argument, NULL, 'e'},

{"right\_down", required\_argument, NULL, 'r'},

{"exchange\_type", required\_argument, NULL, 'T'},

{"freq\_color", no\_argument, NULL, 'F'},

{NULL, no\_argument, NULL, 0}

};

const char\* patternColor = "[0-9]+\\.[0-9]+\\.[0-9]+";

const char\* patternCoords = "[0-9-]+\\.[0-9-]+";

const char\* patternNum = "[0-9-]+";

void callHelp();

void callInfo(char\* fileName, BitmapFileHeader\* BMPFile, BitmapInfoHeader\* BMPnfo);

void callError(int status);

void getKeys(int argc, char\* argv[]);

void parseCoords(char\* coords, int\* x, int\* y);

char\* getStrClone(const char\* src);

int matchRegExp(const char\* buffer, const char\* pattern);

void strToNum(char\* num, int\* val);

void strToRGB(char\* str, Rgb\* color);

void print\_file\_header(const BitmapFileHeader header);

void print\_info\_header(const BitmapInfoHeader header);

unsigned int lenRow(const unsigned int width);

unsigned int padding(const unsigned int width);

Rgb\*\* readFile(const char\* fileName, BitmapFileHeader\* BMPFile, BitmapInfoHeader\* BMPInfo);

void writeFile(const char\* fileName, BitmapFileHeader\* BMPFile, BitmapInfoHeader\* BMPInfo, Rgb\*\* arr);

Rgb\*\* drawSquare(BitmapInfoHeader\* BMPInfo, Rgb\*\* arr, SquareProperty squareProperty);

void drawLineByBrezenhem(Rgb\*\* pixels, int x0, int y0, int x1, int y1, BitmapInfoHeader\* bmih, int thick, Rgb color);

void swap(int\* first, int\* second);

void drawCircle(int x, int y, int radius, Rgb color, Rgb\*\* pixels, int width, int height);

void drawPoint(int x0, int y0, Rgb color, Rgb\*\* pixels, int width, int height);

void fillingSquare(BitmapInfoHeader\* BMPInfo, Rgb\*\* arr, SquareProperty squareProperty);

Rgb\*\* exchangeAreas(BitmapInfoHeader\* BMPinfo, char\* typeExchange, int left, int up, int right, int down, Rgb\*\* pixels);

void checkCorrectPlacement(int\* left, int\* up, int\* right, int\* down, int width, int height);

void validCoord(int\* value, const int edge);

int isCorrectType(char\* type);

void copyArea(Rgb\*\* areas, const int widthArea, const int heightArea, int left, int up, Rgb\*\* pixels);

void pasteArea(Rgb\*\* area, Rgb\*\* pixels, const int width, const int height, const int left, const int up);

Rgb\*\* findAndReplaceColor(BitmapInfoHeader \*BMPInfo, Rgb\*\* pixels, Rgb color, char\* img);

int isColorEqual(Rgb firColor, Rgb secColor);

int writeInMap(Map\* freqColor, Rgb color);

int findMaxInMap(Map\* freqColor);

Rgb bitfox\_get\_primecolor\_direct(char \*FILE\_NAME);

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

{

if(argc == 1)

callHelp();

else

getKeys(argc, argv);

return 0;

}

void callError(const int status){

switch (status)

{

case 40:

fprintf(stderr, "Incorrect color value.\n");

break;

case 41:

fprintf(stderr, "Incorrect type flags.\n");

break;

case 42:

fprintf(stderr, "Incorrect count of flags.\n");

break;

case 43:

fprintf(stderr, "Incorrect fill/fill\_color flags.\n");

break;

case 44:

fprintf(stderr, "Input and Output files is the same.\n");

break;

case 45:

fprintf(stderr, "Flag argument does not match pattern.\n");

break;

case 46:

fprintf(stderr, "Inpossible open the file.\n");

break;

case 47:

fprintf(stderr, "Your area is too small.\n");

break;

case 48:

fprintf(stderr, "file header not match.\n");

default:

fprintf(stderr, "Unknown error...\n");

break;

}

printf("call --help or -h to see more information.\n");

exit(status);

}

void callHelp(){

printf("Course work for option 4.4, created by Ryabov Mikhail.\n");

exit(0);

}

void callInfo(char\* fileName, BitmapFileHeader\* BMPFile, BitmapInfoHeader\* BMPnfo){

printf("Name BMP image: %s\n", fileName);

print\_file\_header(\*BMPFile);

print\_info\_header(\*BMPnfo);

exit(0);

}

int matchRegExp(const char\* buffer, const char\* pattern){

regex\_t regexCompiled;

regcomp(&regexCompiled, pattern, REG\_EXTENDED);

if(regexec(&regexCompiled, buffer, 0, NULL, 0) == 0)

return 1;

return 0;

}

void getKeys(const int argc, char\* argv[])

{

opterr = 0;

int optIndex;

int isCallInfo = 0;

int isSquareFunc = 0, isExchangeFunc = 0, isFreqColorFunc = 0;

int left, up, isLeftUp = 0;

int right, down, isRightDown = 0;

int sideSize, isSideSize = 0;

int thickness, isThickness = 0;

Rgb color;

int isColor = 0;

int isFill = 0;

Rgb fillColor;

int isFillColor = 0;

int isTypeExchange = 0;

char\* inputImage = NULL;

char\* outputImage = NULL;

char\* typeExchange = NULL;

int key = getopt\_long(argc, argv, "Ii:o:hSl:s:t:C:fc:er:T:F", main\_keys, &optIndex);

while(key != -1)

{

//printf("key - %c\n", key);

switch (key)

{

case 'I':

isCallInfo = 1;

break;

case 'i':

inputImage = optarg;

break;

case 'o':

outputImage = optarg;

break;

case 'h':

callHelp();

break;

case 'S':

isSquareFunc = 1;

break;

case 'l':

isLeftUp = 1;

parseCoords(optarg, &left, &up);

break;

case 's':

isSideSize = 1;

strToNum(optarg, &sideSize);

break;

case 't':

isThickness = 1;

strToNum(optarg, &thickness);

break;

case 'C':

isColor = 1;

strToRGB(optarg, &color);

break;

case 'f':

isFill = 1;

break;

case 'c':

isFillColor = 1;

strToRGB(optarg, &fillColor);

break;

case 'e':

isExchangeFunc = 1;

break;

case 'r':

isRightDown = 1;

parseCoords(optarg, &right, &down);

break;

case 'T':

isTypeExchange = 1;

typeExchange = optarg;

break;

case 'F':

isFreqColorFunc = 1;

break;

case '?':

default:

callError(41);

}

key = getopt\_long(argc, argv, "Ii:o:hSl:s:t:C:fc:er:T:F", main\_keys, &optIndex);

}

if(inputImage == NULL)

inputImage = argv[argc - 1];

BitmapFileHeader\* BMPFile = (BitmapFileHeader\*)malloc(sizeof(BitmapFileHeader));

BitmapInfoHeader\* BMPInfo = (BitmapInfoHeader\*)malloc(sizeof(BitmapInfoHeader));

Rgb\*\* pixels = readFile(inputImage, BMPFile, BMPInfo);

if(outputImage == NULL)

outputImage = "out.bmp";

if(!strcmp(inputImage, outputImage))

callError(44);

if(isCallInfo)

callInfo(inputImage, BMPFile, BMPInfo);

const int funcSum = isSquareFunc + isExchangeFunc + isFreqColorFunc;

if(funcSum != 1)

callError(42);

Rgb\*\* newPixels = pixels;

int sumStatusKeys;

int sumOtherKeys;

if(isSquareFunc)

{

sumStatusKeys = isLeftUp + isSideSize + isThickness + isColor;

sumOtherKeys = isRightDown + isTypeExchange;

if(sumOtherKeys != 0 || sumStatusKeys != 4)

callError(42);

if(isFill == 1 && isFillColor == 0)

callError(43);

if(isFill == 1 && isFillColor == 1)

sumStatusKeys += 2;

const SquareProperty squareProperty = {sumStatusKeys, left, up, sideSize, thickness, color, fillColor};

newPixels = drawSquare(BMPInfo, pixels, squareProperty);

}

if(isExchangeFunc)

{

sumStatusKeys = isLeftUp + isRightDown + isTypeExchange;

sumOtherKeys = isSideSize + isThickness + isColor + isFill + isFillColor;

if(sumOtherKeys != 0 || sumStatusKeys != 3)

callError(42);

newPixels = exchangeAreas(BMPInfo, typeExchange, left, up, right, down, pixels);

}

if(isFreqColorFunc)

{

sumStatusKeys = isColor;

sumOtherKeys = isLeftUp + isRightDown + isSideSize + isThickness + isFillColor + isFill + isTypeExchange;

if(sumOtherKeys != 0 || sumStatusKeys != 1)

callError(42);

newPixels = findAndReplaceColor(BMPInfo, pixels, color, inputImage);

}

writeFile(outputImage, BMPFile, BMPInfo, newPixels);

}

Rgb bitfox\_get\_primecolor\_direct(char \*FILE\_NAME)

{

Rgb primecolor = {0, 0, 0};

BYTE hdr[54];

dword \*counts;

dword max\_count = 0;

word w, h;

word i, j;

FILE\* fp = fopen(FILE\_NAME, "rb");

counts = calloc(256 \* 256 \* 256, sizeof(\*counts));

fread(hdr, sizeof(hdr), 1, fp);

w = (hdr[19] << 8) | hdr[18];

h = (hdr[23] << 8) | hdr[22];

for (i = 0; i < h; i++) {

for (j = 0; j < w; j++) {

Rgb rgb;

dword idx;

if (fread(&rgb, 3, 1, fp) < 1) {

fprintf(stderr, "Unexpected end of file.\n");

exit(48);

}

idx = (rgb.r << 16) | (rgb.g << 8) | rgb.b;

if (++counts[idx] > max\_count) {

max\_count = counts[idx];

primecolor = rgb;

}

}

j = 3 \* w;

while (j++ % 4) getc(fp);

}

free(counts);

fclose(fp);

return primecolor;

}

Rgb\*\* findAndReplaceColor(BitmapInfoHeader \*BMPInfo, Rgb\*\* pixels, Rgb color, char\* img){

// Map\* freqColor = (Map\*)malloc(sizeof(Map));

// freqColor->color = (Rgb\*)malloc(sizeof(Rgb)\*1000);

// freqColor->count = (int\*)malloc(sizeof(int)\*1000);

// freqColor->selfSize = 0;

// freqColor->capacity = 1000;

const int width = BMPInfo->width;

const int height = BMPInfo->height;

// for(int y = 0; y < height; y++){

// for(int x = 0; x < width; x++) {

// writeInMap(freqColor, pixels[y][x]);

// }

// }

// int indexMax = findMaxInMap(freqColor);

//printf("indexMax - %d count - %d color - %d %d %d\n", indexMax, freqColor->count[indexMax], freqColor->color[indexMax].r,freqColor->color[indexMax].g, freqColor->color[indexMax].b);

Rgb primeCol = bitfox\_get\_primecolor\_direct(img);

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

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

if(isColorEqual(pixels[y][x], primeCol)){

pixels[y][x].r = color.r;

pixels[y][x].g = color.g;

pixels[y][x].b = color.b;

}

}

}

return pixels;

}

int findMaxInMap(Map\* freqColor){

int index = 0;

int max = 0;

for(int i = 0; i < freqColor->selfSize; i++){

if(freqColor->count[i] > max){

max = freqColor->count[i];

index = i;

}

}

return index;

}

int writeInMap(Map\* freqColor, Rgb color){

if(freqColor->selfSize >= freqColor->capacity){

freqColor->capacity \*= 2;

freqColor->color = (Rgb\*)realloc(freqColor->color,sizeof(Rgb) \* freqColor->capacity);

freqColor->count = (int\*)realloc(freqColor->count, sizeof(int) \* freqColor->capacity);

}

if(!freqColor->selfSize){

freqColor->color[freqColor->selfSize] = color;

freqColor->count[freqColor->selfSize++] = 0;

return 1;

}

int flag = 0;

for(int i = 0; i < freqColor->selfSize; i++){

if(isColorEqual(freqColor->color[i], color)){

flag = 1;

freqColor->count[i] += 1;

}

}

if(!flag){

freqColor->color[freqColor->selfSize] = color;

freqColor->count[freqColor->selfSize++] = 0;

}

return 1;

}

int isColorEqual(Rgb firColor, Rgb secColor){

if(firColor.r == secColor.r && firColor.g == secColor.g && firColor.b == secColor.b)

return 1;

return 0;

}

Rgb\*\* exchangeAreas(BitmapInfoHeader\* BMPinfo, char\* typeExchange, int left, int up, int right, int down, Rgb\*\* pixels)

{

checkCorrectPlacement(&left, &up, &right, &down, BMPinfo->width, BMPinfo->height);

if(abs(right - left) <= 1 || abs(down - up) <= 1)

return pixels;

const int widthArea = (int)(right - left)/2;

const int heightArea = (int)(down - up)/2;

const int type = isCorrectType(typeExchange);

if(!type) callError(45);

Rgb\*\* leftUp = (Rgb\*\*)malloc(sizeof(Rgb\*\*) \* heightArea);

Rgb\*\* leftDown = (Rgb\*\*)malloc(sizeof(Rgb\*\*) \* heightArea);

Rgb\*\* rightUp = (Rgb\*\*)malloc(sizeof(Rgb\*\*) \* heightArea);

Rgb\*\* rightDown = (Rgb\*\*)malloc(sizeof(Rgb\*\*) \* heightArea);

copyArea(leftUp, widthArea, heightArea, left, up, pixels);

copyArea(leftDown, widthArea, heightArea, left, up+heightArea, pixels);

copyArea(rightUp, widthArea, heightArea, left + widthArea, up, pixels);

copyArea(rightDown, widthArea, heightArea, left + widthArea, up+heightArea, pixels);

if(type == 2){

pasteArea(leftUp, pixels, widthArea, heightArea, left, up+heightArea);

pasteArea(leftDown, pixels, widthArea, heightArea, left+widthArea, up+heightArea);

pasteArea(rightUp, pixels, widthArea, heightArea, left, up);

pasteArea(rightDown, pixels, widthArea, heightArea, left+widthArea, up);

}

if(type == 1){

pasteArea(leftUp, pixels, widthArea, heightArea, left+widthArea, up);

pasteArea(leftDown, pixels, widthArea, heightArea, left, up);

pasteArea(rightUp, pixels, widthArea, heightArea, left+widthArea, up+heightArea);

pasteArea(rightDown, pixels, widthArea, heightArea, left, up+heightArea);

}

if(type == 3){

pasteArea(leftUp, pixels, widthArea, heightArea, left+widthArea, up+heightArea);

pasteArea(leftDown, pixels, widthArea, heightArea, left+widthArea, up);

pasteArea(rightUp, pixels, widthArea, heightArea, left, up+heightArea);

pasteArea(rightDown, pixels, widthArea, heightArea, left, up);

}

return pixels;

}

void pasteArea(Rgb\*\* area, Rgb\*\* pixels, const int width, const int height, const int left, const int up)

{

int y = 0;

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

int x = 0;

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

pixels[i][j] = area[y][x];

x++;

}

y++;

}

}

void copyArea(Rgb\*\* area, const int widthArea, const int heightArea, const int left, const int up, Rgb\*\* pixels)

{

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

area[i] = (Rgb\*)malloc(sizeof(Rgb) \* widthArea);

int y = 0, x;

for(int i = up; i < up + heightArea; i++){

x = 0;

for(int j = left; j < left + widthArea; j++){

//printf("x - %d y - %d i - %d j - %d\n", x, y, i, j);

area[y][x] = pixels[i][j];

x++;

}

y++;

}

}

int isCorrectType(char\* type)

{

if(!strcmp("clockwise", type)) return 1;

if(!strcmp("counterclockwise", type)) return 2;

if(!strcmp("diagonals", type)) return 3;

return 0;

}

void checkCorrectPlacement(int\* left, int\* up, int\* right, int\* down, int width, int height)

{

validCoord(left, width);

validCoord(right, width);

validCoord(up, height);

validCoord(down, height);

//if(abs((\*right) - (\*left)) <= 1 || abs((\*up) - (\*down)) <= 1 ) callError(47);

if(\*left > \*right)

{

int buf = \*left;

\*left = \*right;

\*right = buf;

}

if(\*down < \*up)

{

int buf = \*up;

\*up = \*down;

\*down = buf;

}

}

void validCoord(int\* value, const int edge)

{

(\*value) < 0 ? (\*value) = 0 : (\*value);

(\*value) >= edge ? (\*value) = edge - 1 : (\*value);

}

Rgb\*\* drawSquare(BitmapInfoHeader\* BMPInfo, Rgb\*\* arr, SquareProperty sp)

{

if(sp.sumStatusKeys == 6)

{

fillingSquare(BMPInfo, arr, sp);

drawLineByBrezenhem(arr, sp.left, sp.up, sp.left+sp.side\_size, sp.up, BMPInfo, sp.thickness, sp.color);

drawLineByBrezenhem(arr, sp.left+sp.side\_size, sp.up, sp.left+sp.side\_size, sp.up+sp.side\_size, BMPInfo, sp.thickness, sp.color);

drawLineByBrezenhem(arr, sp.left, sp.up+sp.side\_size, sp.left+sp.side\_size, sp.up+sp.side\_size, BMPInfo, sp.thickness, sp.color);

drawLineByBrezenhem(arr, sp.left, sp.up, sp.left, sp.up+sp.side\_size, BMPInfo, sp.thickness, sp.color);

}

else

{

drawLineByBrezenhem(arr, sp.left, sp.up, sp.left+sp.side\_size, sp.up, BMPInfo, sp.thickness, sp.color);

drawLineByBrezenhem(arr, sp.left+sp.side\_size, sp.up, sp.left+sp.side\_size, sp.up+sp.side\_size, BMPInfo, sp.thickness, sp.color);

drawLineByBrezenhem(arr, sp.left, sp.up+sp.side\_size, sp.left+sp.side\_size, sp.up+sp.side\_size, BMPInfo, sp.thickness, sp.color);

drawLineByBrezenhem(arr, sp.left, sp.up, sp.left, sp.up+sp.side\_size, BMPInfo, sp.thickness, sp.color);

}

return arr;

}

void fillingSquare(BitmapInfoHeader\* BMPInfo, Rgb\*\* arr, SquareProperty sp)

{

const int left = sp.left;

const int side = sp.left + sp.side\_size;

const int up = sp.up;

const int sideUp = sp.up + sp.side\_size;

for(int i = left; i < side; i++)

{

for(int j = up; j < sideUp; j++)

{

drawPoint(i, j, sp.fill\_color, arr, BMPInfo->width, BMPInfo->height);

}

}

}

void drawLineByBrezenhem(Rgb\*\* pixels, int x0, int y0, int x1, int y1, BitmapInfoHeader\* bifh, int thick, Rgb color)

{

int radius;// = (int)thick/2;

if(thick == 1) radius = 0;

else if (thick % 2 == 1) radius = (int)(thick+1)/2;

else radius = (int)thick/2;

const int deltaX = abs(x1 - x0);

const int deltaY = abs(y1 - y0);

const int signX = x0 < x1 ? 1 : -1;

const int signY = y0 < y1 ? 1 : -1;

int error = deltaX - deltaY;

//drawPoint(x1, y1, color, pixels, bifh->width, bifh->height);

drawCircle(x1, y1, radius, color, pixels, bifh->width, bifh->height);

while(x0 != x1 || y0 != y1)

{

//drawPoint(x0, y0, color, pixels, bifh->width, bifh->height);

drawCircle(x0, y0, radius, color, pixels, bifh->width, bifh->height);

int error2 = error \* 2;

if(error2 > -deltaY)

{

error -= deltaY;

x0 += signX;

}

if(error2 < deltaX)

{

error += deltaX;

y0 += signY;

}

}

}

void drawCircle(int x0, int y0, int radius, Rgb color, Rgb\*\* pixels, int width, int height)

{

int x = radius;

int y = 0;

int radiusError = 1 - x;

while (x >= y)

{

drawPoint(x + x0, y + y0, color, pixels, width, height);

drawPoint(y + x0, x + y0, color, pixels, width, height);

drawPoint(-x + x0, y + y0, color, pixels, width, height);

drawPoint(-y + x0, x + y0, color, pixels, width, height);

drawPoint(-x + x0, -y + y0, color, pixels, width, height);

drawPoint(-y + x0, -x + y0, color, pixels, width, height);

drawPoint(x + x0, -y + y0, color, pixels, width, height);

drawPoint(y + x0, -x + y0, color, pixels, width, height);

y++;

if (radiusError < 0)

{

radiusError += 2 \* y + 1;

}

else

{

x--;

radiusError += 2 \* (y - x + 1);

}

}

for(int cury=-radius; cury<=radius; cury++)

for(int curx=-radius; curx<=radius; curx++)

if(curx\*curx+cury\*cury <= radius\*radius)

drawPoint(x0+curx, y0+cury, color, pixels, width, height);

}

void drawPoint(int x0, int y0, Rgb color, Rgb\*\* pixels, int width, int height)

{

if(x0 >= 0 && x0 < width && y0 >= 0 && y0 < height)

{

pixels[y0][x0].r = color.r;

pixels[y0][x0].g = color.g;

pixels[y0][x0].b = color.b;

}

}

void swap(int\* first, int\* second)

{

int\* buffer = first;

first = second;

second = buffer;

}

Rgb\*\* readFile(const char\* fileName, BitmapFileHeader\* BMPFile, BitmapInfoHeader\* BMPInfo)

{

FILE \*file = fopen(fileName, "rb");

if(file == NULL)

callError(46);

fread(BMPFile, 1, sizeof(BitmapFileHeader), file);

fread(BMPInfo, 1, sizeof(BitmapInfoHeader), file);

if (BMPFile->signature != 0x4d42){

fprintf(stderr, "File Error: file must be bmp\n");

fclose(file);

exit(48);

}

if (BMPInfo->compression != 0){

fprintf(stderr, "File Error: file must be uncompressed\n");

fclose(file);

exit(48);

}

if (BMPInfo->bitsPerPixel != 24){

fprintf(stderr, "File Error: file must have 24 bits per pixel\n");

fclose(file);

exit(48);

}

if (BMPInfo->width > 50000 || BMPInfo->height > 50000){

fprintf(stderr, "file is too big\n");

fclose(file);

exit(48);

}

unsigned int width = BMPInfo->width;

unsigned int height = BMPInfo->height;

Rgb\*\* bitmap = (Rgb\*\*)malloc(sizeof(Rgb\*) \* height);

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

bitmap[i] = (Rgb\*)malloc(lenRow(width));

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

fread(bitmap[height - i - 1], 1, lenRow(width), file);

fclose(file);

return bitmap;

}

void writeFile(const char\* fileName, BitmapFileHeader\* BMPFile, BitmapInfoHeader\* BMPInfo, Rgb\*\* arr)

{

FILE\* file = fopen(fileName, "wb");

fwrite(BMPFile, 1, sizeof(BitmapFileHeader), file);

fwrite(BMPInfo, 1, sizeof(BitmapInfoHeader), file);

const unsigned int width = BMPInfo->width;

const unsigned int height = BMPInfo->height;

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

fwrite(arr[height - i - 1], 1, lenRow(width), file);

fclose(file);

}

unsigned int padding(const unsigned int width)

{

unsigned int padding = (width \* sizeof(Rgb)) % 4;

if(padding) padding = 4 - padding;

return padding;

}

unsigned int lenRow(const unsigned int width)

{

return width \* sizeof(Rgb) + padding(width);

}

void parseCoords(char\* coords, int\* x, int\* y){

if(!matchRegExp(coords, patternCoords))

callError(45);

char\* argcpy = getStrClone(coords);

(\*x) = atoi(strtok(argcpy, "."));

(\*y) = atoi(strtok(NULL, "."));

}

void strToNum(char\* num, int\* val){

if(!matchRegExp(num, patternNum))

callError(43);

char\* argcpy = getStrClone(num);

(\*val) = atoi(strtok(argcpy, "."));

if((\*val) <= 0)

callError(43);

}

void strToRGB(char\* str, Rgb\* color){

if(!matchRegExp(str, patternColor))

callError(40);

char\* argcpy = getStrClone(str);

const int r = atoi(strtok(argcpy, "."));

const int g = atoi(strtok(NULL, "."));

const int b = atoi(strtok(NULL, "."));

if((r > 255 || r < 0) || (g > 255 || g < 0) || (b > 255 || b < 0))

callError(40);

color->r = r;

color->g = g;

color->b = b;

}

char\* getStrClone(const char\* src){

char\* strClone = (char\*)malloc(sizeof(char)\*(strlen(src) + 1));

strcpy(strClone, src);

strClone[strlen(src)] = '\0';

return strClone;

}

void print\_file\_header(const BitmapFileHeader header){

printf("signature:\t%x (%hu)\n", header.signature, header.signature);

printf("filesize:\t%x (%u)\n", header.filesize, header.filesize);

printf("reserved1:\t%x (%hu)\n", header.reserved1, header.reserved1);

printf("reserved2:\t%x (%hu)\n", header.reserved2, header.reserved2);

printf("pixelArrOffset:\t%x (%u)\n", header.pixelArrOffset, header.pixelArrOffset);

}

void print\_info\_header(const BitmapInfoHeader header){

printf("headerSize:\t%x (%u)\n", header.headerSize, header.headerSize);

printf("width: \t%x (%u)\n", header.width, header.width);

printf("height: \t%x (%u)\n", header.height, header.height);

printf("planes: \t%x (%hu)\n", header.planes, header.planes);

printf("bitsPerPixel:\t%x (%hu)\n", header.bitsPerPixel, header.bitsPerPixel);

printf("compression:\t%x (%u)\n", header.compression, header.compression);

printf("imageSize:\t%x (%u)\n", header.imageSize, header.imageSize);

printf("xPixelsPerMeter:\t%x (%u)\n", header.xPixelsPerMeter, header.xPixelsPerMeter);

printf("yPixelsPerMeter:\t%x (%u)\n", header.yPixelsPerMeter, header.yPixelsPerMeter);

printf("colorsInColorTable:\t%x (%u)\n", header.colorsInColorTable, header.colorsInColorTable);

printf("importantColorCount:\t%x (%u)\n", header.importantColorCount, header.importantColorCount);

}

**приложение б**

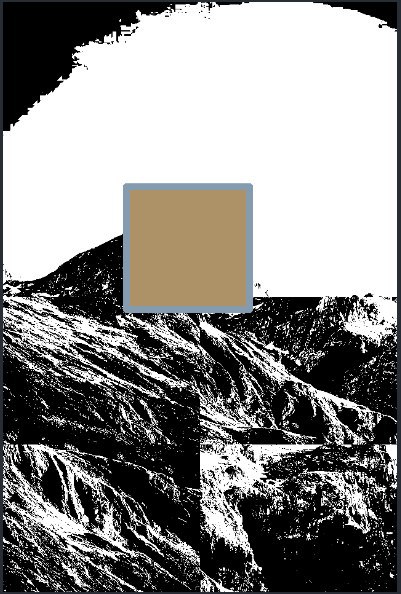
**Тестирование**

Рисование квадрата с диагоналями: ./main --square --thickness 10 --side\_size 200 --left\_up 200.300 --color 133.155.176 --fill --fill\_color 173.145.103 --input ./4.bmp

Рисунок 1. Входное изображение



Рисунок 2. Выходное изображение



Поменять местами 4 куска области: ./main --exchange --input ./out.bmp --exchange\_type clockwise --left\_up -100.-100 --right\_down 1000.1000 --output ./kek.bmp

Рисунок 3. Входное изображение

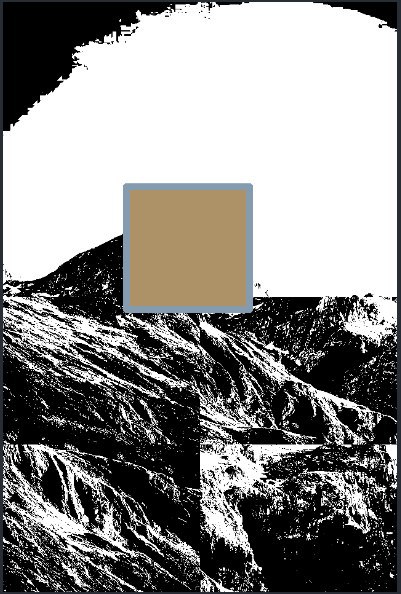
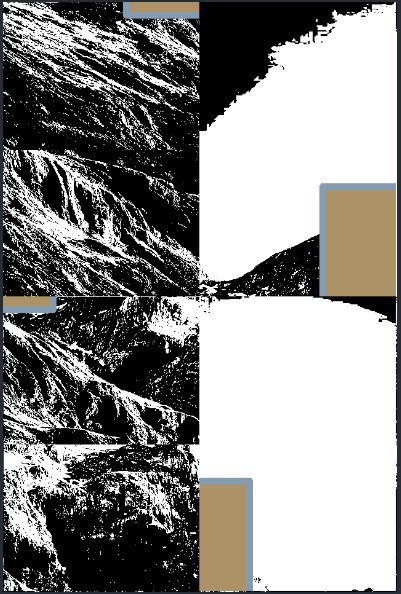


Рисунок 4. Выходное изображение



Поворот изображения: ./main --freq\_color --color 134.245.121 --input ./kek.bmp

Рисунок 5. Входное изображение

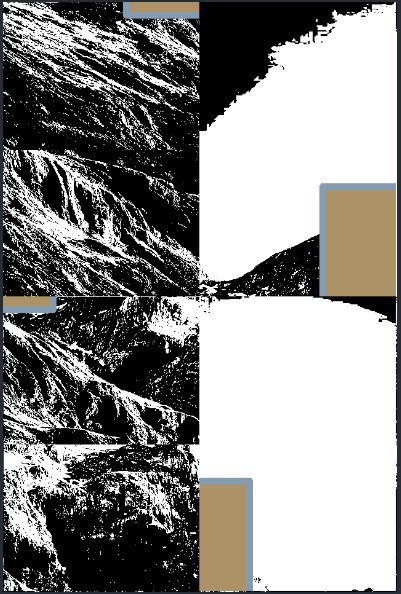


Рисунок 6. Выходное изображение

