МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

Курсовая РАБОТА

по дисциплине «Программирование»

Тема: Обработка изображений в формате BMP.

Студент гр. 0382 _____ Довченко М.К.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2021

Студент Довченко М.К.

Группа 0382

Тема работы: Обработка изображений в формате BMP.

Исходные данные: программу требуется реализовать в виде терминального интерфейса. Программа должна принимать аргументы, давать возможность сохранить измененную картинку в другой файл, выполнять функционал по обработке изображений, указанный в задании.

Содержание пояснительной записки:

«Содержание», «Введение», «Задание работы», «Ход выполнения работы»

«Заключение», «Список использованных источников».

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Дата выдачи задания: 05.04.2021

Дата сдачи реферата:

Дата защиты реферата:

Студент		Довченко М.К.
Преподаватель		Берленко Т.А.

АННОТАЦИЯ

В ходе выполнения курсовой работы была написана программа, которая редактирует и сохраняет изображение в формате BMP. Обработка осуществляется в соответствии с заданием. Функционал определяется фильтром RGB-компонента, рисованием квадрата, сменой местами четырех областей картинки, нахождением самого часто встречаемого цвета и изменением его.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы

Написать на языке Си программу, которая реализует различные функции для обработки изображений формата BMP.

Основные задачи

Реализация консольного интерфейса с использованием getopt_long.

Обеспечение стабильной работы с различными форматами BMP, обработка исключительных случаев.

Реализация сохранения изображения в новом файле.

Методы решения

Разработка программы велась с помощью операционной системы Linux и с использованием приложения Virtual Studio Code.

1. ЗАДАНИЕ

Программа **должна** иметь CLI или GUI. Более подробно тут: **http://se.moevm.info/doku.php/courses:programming:rules_extra_kurs**

Общие сведения

- 24 бита на цвет
- без сжатия
- файл всегда соответствует формату BMP (но стоит помнить, что версий у формата несколько)
- 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 реализовывать весь следующий функционал по обработке bmp-файла
-

- Фильтр rgb-компонент. Этот инструмент должен позволять для всего изображения либо установить в 0 либо установить в 255 значение заданной компоненты. Функционал определяется
    - Какую компоненту требуется изменить
    - В какой значение ее требуется изменить
- Рисование квадрата. Квадрат определяется:

    - Координатами левого верхнего угла
    - Размером стороны
    - Толщиной линий
    - Цветом линий
    - Может быть залит или нет
    - Цветом которым он залит, если пользователем выбран залитый
- Поменять местами 4 куска области. Выбранная пользователем прямоугольная область делится на 4 части и эти части меняются местами. Функционал определяется:
    - Координатами левого верхнего угла области
    - Координатами правого нижнего угла области
    - Способом обмена частей: “по кругу”, по диагонали
- Находит самый часто встречаемый цвет и заменяет его на другой заданный цвет. Функционал определяется
    - Цветом, в который надо перекрасить самый часто встречаемый цвет

2. ХОД ВЫПОЛНЕНИЯ РАБОТЫ

2.1. Структура программы

Вначале прописывается первичное устройство программы. Объявляются заголовочные файлы стандартной библиотеки Си, такие как <stdio.h>, <string.h>, <stdint.h>, <stdlib.h>, а также <getopt.h> для настройки и работы пользовательского интерфейса .

2.2. Считывание картинки

Объявляются структуры, отвечающие за хранение данных об изображении:

Далее происходит считывание данных с помощью функции fread(), принимающей адрес структуры, в которую надо записать данные, ее размер, кол-во считываемых полей и сам файл, из которого надо производить считывание.

2.3. Хранение картинки

Для выполнения данной задачи создается структура BMP состоящая из ширины, высоты, и структуры RGB хранящей цвет пикселей. Эта структура состоит из 3-х элементов, отвечающих за хранение синего, зеленого и красного цвета.

2.4. Создание функции для взаимодействия с пользователем

Когда функция main() вызывается кодом инициализации программы, обработка команд, передающихся в этот метод, уже выполнена. Параметр argc содержит число, соответствующее числу параметров или аргументов, и argv содержит

массив указателей на эти аргументы. Под аргументами библиотека подразумевает название программы и все остальные параметры, что следуют за именем команды и отделяются друг от друга пробелом.

Для работы функции getopt_long был реализован цикл while, объявленный как while((opt = getopt_long(argc, argv, “f:s:w:c:ih”, long_options, &option_index)) != -1)

2.5. Создание функций обработки изображения

2.5.1. Создание функции для реализации фильтра rgb-компонента.

При вводе ключа -f или --filter программе требуется передать 2 аргумента: colour_option и colour_value, первый аргумент отвечает за выбранный цвет который нужно изменить, второй аргумент отвечает за значение выбранного цвета. В зависимости от выбора цвета вызывается одна из функций для изменения цвета: changeRed, changeGreen, changeBlue.

2.5.2. Создание функции для рисования квадрата

Для того чтобы нарисовать квадрат были использованы такие функции как drawPixel, изменяющая цвет пикселя на заданный, drawLine, рисующая линию от заданной точки до заданной точки и функция drawSquare, которая и рисует сам квадрат. Функция принимает структуру BMP для изменения картинки, координаты верхнего левого угла, длину стороны квадрата, толщину стороны квадрата, цвет линий квадрата, переменную fill отвечающую за выбор опции заливки (квадрат “заливается” только если fill = 1), цвет заливки.

Сначала функция проверяют являются ли допустимыми некоторые переменные чтобы избежать падения программы. Далее функция сравнивает переменную fill с единицей чтобы понять нужно ли выполнять заливку. Функция рисует квадрат с помощью 4х циклов содержащих функцию drawLine,

каждый из циклов выполняется количество раз равной переменной thickness, отвечающей за толщину стороны.

2.5.3. Создание функции для смены четырех областей картинки.

Функция swapFour принимает структуру BMP для изменения картинки, имя картинки чтобы создать копию картинки для дальнейшего использования, координаты верхнего левого и нижнего правого углов, переменную option принимающая значения либо r или d для определения способа смены области картинки (по кругу или по диагонали).

Сначала функция создает копию картинки, затем находится высота и ширина одной области, далее функция проверяет некоторые значения. Далее в зависимости от опции функция поворачивает картинку либо по кругу либо по диагонали.

Для того чтобы повернуть область картинки по кругу функция сначала заменяет нижний левый фрагмент на нижний правый фрагмент, затем нижний правый фрагмент на верхний правый фрагмент, затем верхний правый фрагмент на верхний левый фрагмент. Чтобы заменить верхний левый фрагмент используется нижний правый фрагмент из копии изображения.

Для того чтобы повернуть область картинки по диагонали сначала меняются местами нижний правый и верхний левый фрагменты. Для того чтобы не использовать копию изображения пиксели заменяются следующим образом: переменной t присваивается значение цвета пикселя нижнего правого фрагмента, затем значению цвета пикселя нижнего правого фрагмента присваивается значение цвета пикселя верхнего левого фрагмента, затем значению цвета пикселя верхнего левого фрагмента присваивается значение переменной t. Тот же алгоритм используется для того чтобы поменять местами верхний правый и нижний левый фрагменты.

2.5.4 Создание функции для изменения самого встречающегося цвета.

Функции changeMostFrequentColour передается структура BMP содержащая изображение и набор значений для красного, зеленого и синего цветов на которые нужно изменить значения самого встречающегося цвета.

Сначала инициализируется трехмерный массив чисел 256 на 256 на 256, созданный для хранения какого либо цвета, на него также динамически выделяется память. Далее используя вложенные циклы, который сравнивает значения массива и каждого пикселя, находится самый часто встречающийся цвет. Если цвет уже встречался то к счетчику прибавляется единица и затем массив сравнивается со следующим цветом. Затем после нахождения самого встречающегося цвета в еще одном цикле который сравнивает все пиксели с самым встречающимся цветом происходит замена цветов этого пикселя на переданные в функцию цвета. В конце функции память ранее выделенная на массив содержащий наиболее встречающийся цвет освобождается.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы было создано приложение с консольным интерфейсом для обработки изображений в формате BMP.

Демонстрация работоспособности программы

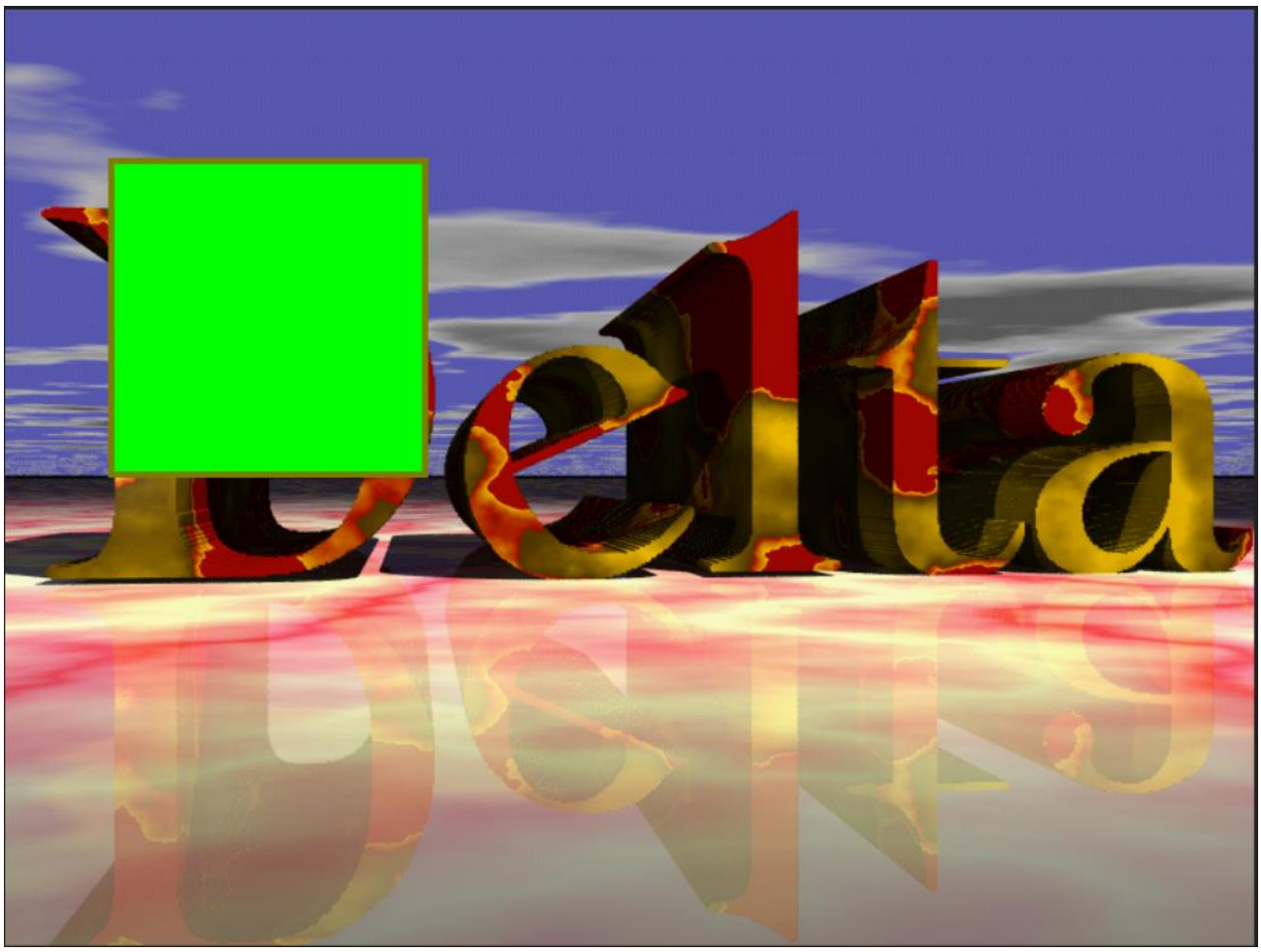
Исходное изображение:

Проверка функции 1:

```
./a.out --filter r,240 RAY.bmp
```

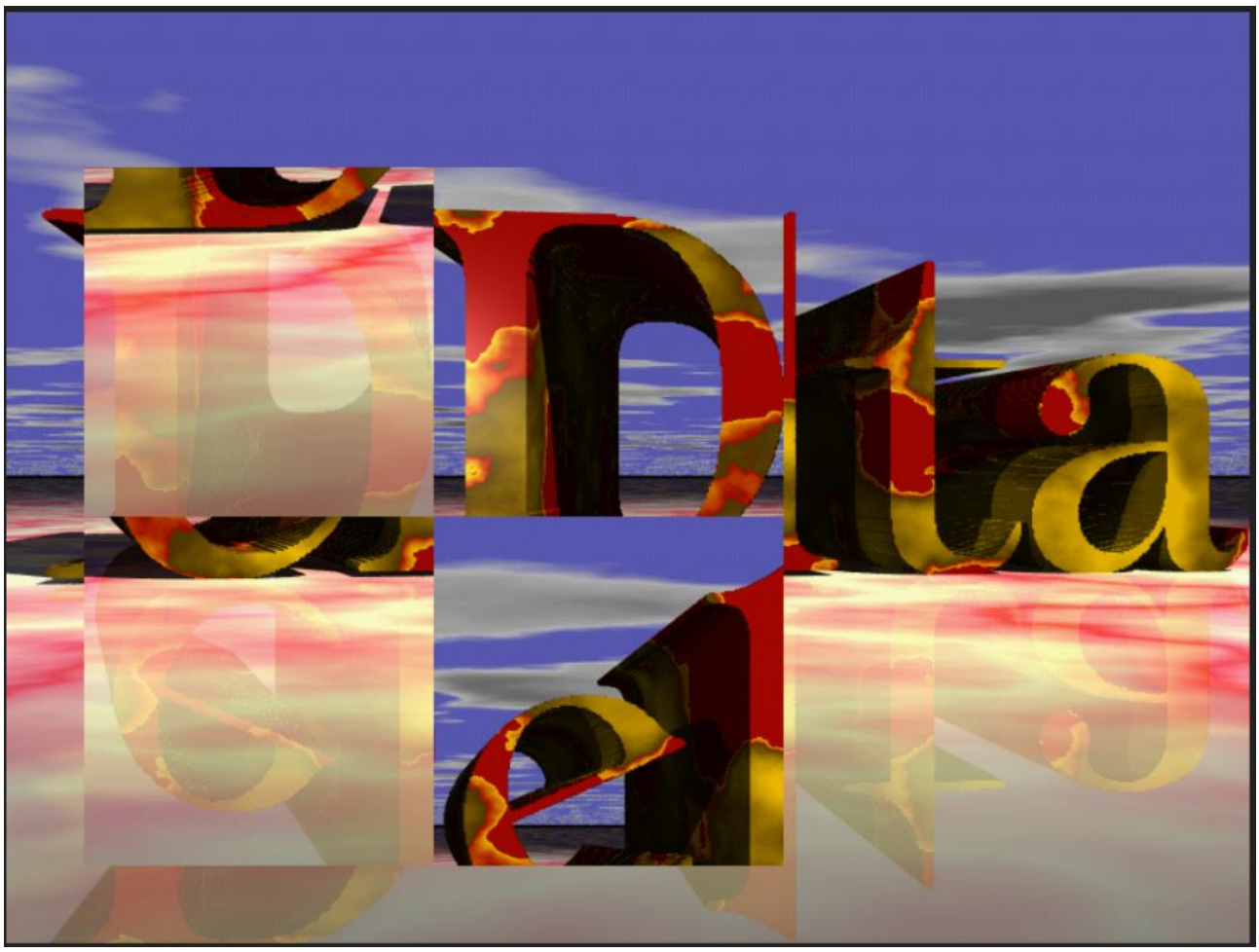
Проверка функции 2:

```
./a.out --square 70,500,200,4,125,125,0,1,0,255,0 RAY.bmp
```

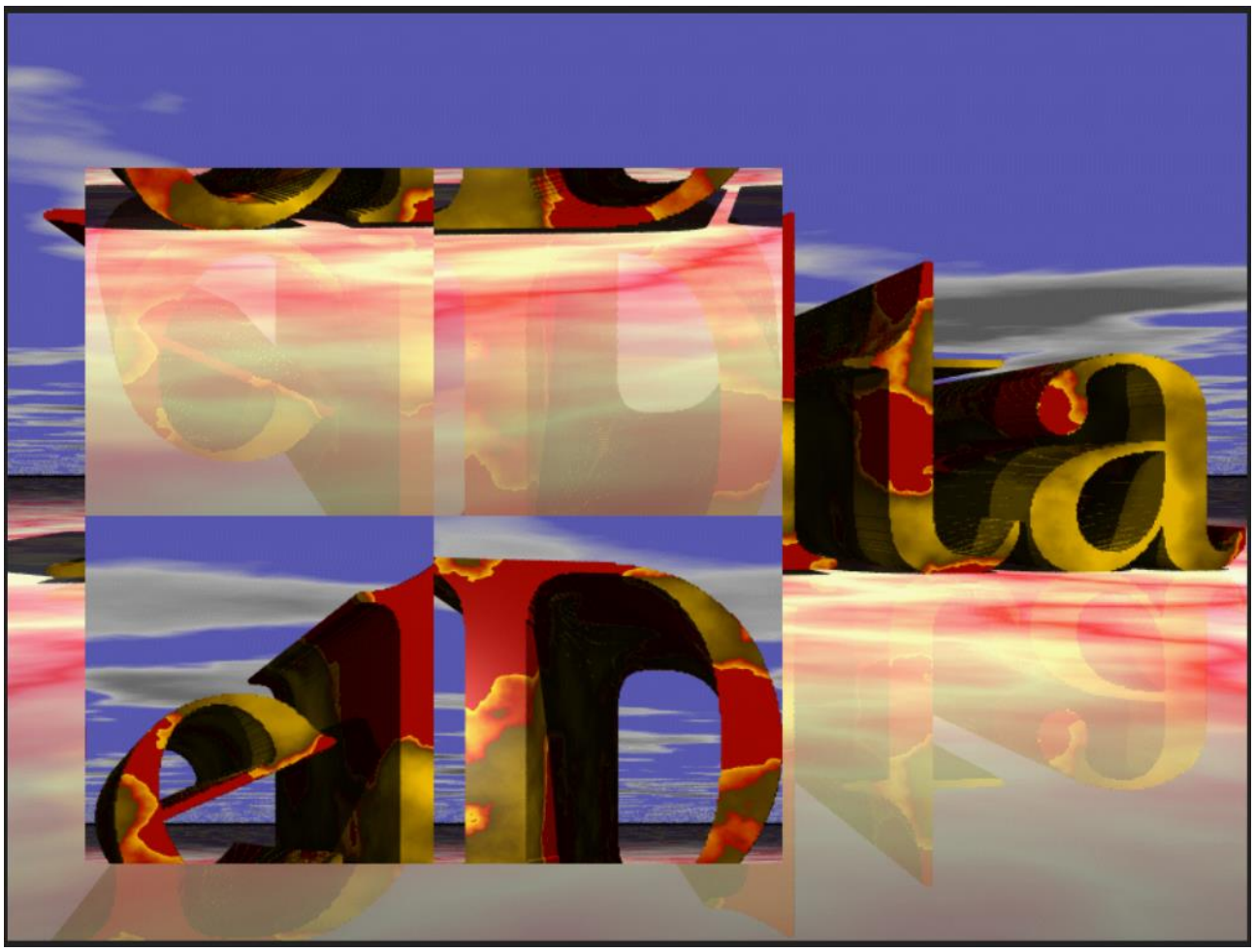


Проверка функции 3:

```
./a.out --swap 50,500,500,50,r RAY.bmp
```



```
./a.out --swap 50,500,500,50,d RAY.bmp
```



Проверка функции 4:

```
./a.out --changefreq 255,0,0 RAY.bmp
```


Delta

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Керниган Б. В., Ритчи Д. М. Язык программирования Си: Пер. с англ. — 3-е изд. — СПб.: Невский Диалект, 2001. — 352 с.

ПРИЛОЖЕНИЕ А.
ИСХОДЫЙ КОД ПРОГРАММЫ

cw.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <getopt.h>
#include <string.h>


#pragma pack (push, 1)
typedef struct
{
    uint16_t signature;
    uint32_t filesize;
    uint16_t reserved1;
    uint16_t reserved2;
    uint32_t pixelArrOffset;
} BitmapFileHeader;


typedef struct
{
    uint32_t headerSize;
    uint32_t width;
    uint32_t height;
```

```
    uint16_t planes;
    uint16_t bitsPerPixel;
    uint32_t compression;
    uint32_t imageSize;
    uint32_t xPixelsPerMeter;
    uint32_t yPixelsPerMeter;
    uint32_t colorsInColorTable;
    uint32_t importantColorCount;
} BitmapInfoHeader;


typedef struct
{
    uint8_t b;
    uint8_t g;
    uint8_t r;
} RGB;


typedef struct
{
    BitmapFileHeader bmfh;
    BitmapInfoHeader bmih;
    RGB **arr;
} BMP;


#pragma pack(pop)
```

```
static uint8_t colour;
static struct option long_options[] = {
    {"filter", required_argument, NULL, 'f'},
    {"square", required_argument, NULL, 's'},
    {"swap", required_argument, NULL, 'w'},
    {"changefreq", required_argument, NULL, 'c'},
    {"info", no_argument, NULL, 'i'},
    {"help", no_argument, NULL, 'h'},
    {NULL, no_argument, NULL, 0}
};

char *INPUTFILE, *OUTPUTFILE = "out.bmp";

void printFileHeader(BitmapFileHeader);
void printInfoHeader(BitmapInfoHeader);
void readImage(const char*, BMP*);
void writeImage(const char*, BMP*);
void changeRed(BMP*, int);
void changeGreen(BMP*, int);
void changeBlue(BMP*, int);
void changeMostFrequentColour(BMP*, int, int, int);
void swapFour(BMP*, const char*, int, int, int, int, char);
void drawSquare(BMP*, int, int, int, int, RGB, int, RGB);
RGB changeColour(uint8_t, uint8_t, uint8_t);
void drawpixel(int ,int, BMP*, RGB);
void drawLine(int, int, int, int, BMP*, RGB);
```

```
int checkPicture(const char*);
void help();
int isNotSupported(BMP*);


int main(int argc, char *argv[]){
    const char* path = "/mnt/c/Users/dovch/uni/c/cw2reg/simp.bmp";
    const char* path_out = "/mnt/c/Users/dovch/uni/c/cw2reg/outsimp.bmp";
    if(argc == 0 || argc == 1){
        help();
        return 0;
    }
    INPUTFILE = malloc(strlen(argv[argc-1]) * sizeof(char));
    strcpy(INPUTFILE, argv[argc-1]);
    BMP bitmap;
    readImage(INPUTFILE, &bitmap);
    if(checkPicture(INPUTFILE)){
        return 0;
    }
    if(isNotSupported(&bitmap)){
        return 0;
    }


    int opt;
    int option_index = 0;
```

```
    while((opt = getopt_long(argc, argv, "f:s:w:c:ih", long_options,
&option_index)) != -1){

        switch (opt)

        {

            case 's': ;

                int x, y, side, thickness, r, g, b, fill, rf, gf, bf;

                int count_arg_s = sscanf(optarg,
"%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d", &x, &y, &side, &thickness, &r,

                       &g, &b, &fill, &rf, &gf, &bf);

                if (count_arg_s < 11) {

                    printf("Введены не все аргументы\n");

                    return 0;

                }

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

                    printf("Цвет должен принимать значения от 0 до
255\n");

                    return 0;

                }

                if((rf > 255) || (rf < 0) || (gf > 255) || (gf < 0) ||
(bf > 255) || (bf < 0)){

                    printf("Цвет должен принимать значения от 0 до
255\n");

                    return 0;

                }

                RGB square_colour;

                RGB fill_colour;

                square_colour.r = r;

                square_colour.g = g;
```

```
                square_colour.b = b;
                fill_colour.r = rf;
                fill_colour.g = gf;
                fill_colour.b = bf;


                drawSquare(&bitmap, x, y, side, thickness, square_colour,
fill, fill_colour);
                break;
            case 'f': ;
                int colour_value;
                char colour_option;
                int count_arg_f = sscanf(optarg, "%c,%d", &colour_option,
&colour_value);
                if (count_arg_f < 2) {
                    printf("Введены не все аргументы\n");
                    return 0;
                }
                if((colour_value > 255) || (colour_value < 0)){
                    printf("Цвет должен принимать значения от 0 до
255\n");
                    return 0;
                }
                switch (colour_option){
                    case 'r':
                        changeRed(&bitmap, colour_value);
                        break;
                    case 'g':
                        changeGreen(&bitmap, colour_value);
```

```
                    break;
                case 'b':
                    changeBlue(&bitmap, colour_value);
                    break;
                default:
                    printf("Указана неправильная опция цвета.\n");
                    break;
                }
                break;
            case 'c': ;
                int r1, g1, b1;
                int count_arg_c = sscanf(optarg, "%d,%d,%d", &r1, &g1,
&b1);
                if (count_arg_c < 3) {
                    printf("Введены не все аргументы\n");
                    return 0;
                }
                if((r1 > 255) || (r1 < 0) || (g1 > 255) || (g1 < 0) ||
(b1 > 255) || (b1 < 0)){
                    printf("Цвет должен принимать значения от 0 до
255\n");
                    return 0;
                }
                changeMostFrequentColour(&bitmap, r1, g1, b1);
                break;
            case 'w': ;
                int w_x1, w_y1, w_x2, w_y2;
                char w_option;
```

```
                int count_arg_w = sscanf(optarg, "%d,%d,%d,%d,%c", &w_x1,
&w_y1, &w_x2, &w_y2, &w_option);
                if (count_arg_w < 5) {
                    printf("Введены не все аргументы\n");
                    return 0;
                }
                swapFour(&bitmap, INPUTFILE, w_x1, w_y1, w_x2, w_y2,
w_option);
                break;
            case 'i':
                printf("\nПодробная информация об изображении\n");
                printFileHeader(bitmap.bmfh);
                printInfoHeader(bitmap.bmih);
                break;
            case 'h':
                help();
                break;
            case '?':
                help();
                break;
            default:
                break;
        }
    }


    writeImage(OUTPUTFILE, &bitmap);
    return 0;
```

```
}



void printFileHeader(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, head-
er.pixelArrOffset);

}

void printInfoHeader(BitmapInfoHeader header){

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

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

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

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

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

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

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

}
```

```
void readImage(const char* path, BMP* bitmap){
    FILE *f = fopen(path, "rb");
    if(!f){
        printf("Не удалось открыть файл\n");
        return;
    }
    fread(&(bitmap->bmfh),1,sizeof(BitmapFileHeader),f);
    fread(&(bitmap->bmih),1,sizeof(BitmapInfoHeader),f);



    bitmap->arr = malloc(bitmap->bmih.height*sizeof(RGB*));
    for(int i=0; i<bitmap->bmih.height; i++){
        bitmap->arr[i] = malloc(bitmap->bmih.width * sizeof(RGB) + (bit-
map->bmih.width*3)%4);
        fread(bitmap->arr[i],1,bitmap->bmih.width * sizeof(RGB) + (bit-
map->bmih.width*3)%4,f);
    }
    fclose(f);
}
void writeImage(const char* path, BMP* bitmap){
    uint32_t H = bitmap->bmih.height;
    uint32_t W = bitmap->bmih.width;
    FILE *ff = fopen(path, "wb");
    fwrite(&(bitmap->bmfh), 1, sizeof(BitmapFileHeader),ff);
    fwrite(&(bitmap->bmih), 1, sizeof(BitmapInfoHeader),ff);
```

```
    for(int i=0; i<H; i++){
        fwrite(bitmap->arr[i],1,W * sizeof(RGB) + (W*3)%4,ff);
    }
    fclose(ff);
}


void changeRed(BMP* bitmap, int n){
    for(int i = 0; i<bitmap->bmih.height; i++){
        for(int j = 0; j<bitmap->bmih.width; j++){
            bitmap->arr[i][j].r = n;
        }
    }
}
void changeGreen(BMP* bitmap, int n){
    for(int i = 0; i<bitmap->bmih.height; i++){
        for(int j = 0; j<bitmap->bmih.width; j++){
            bitmap->arr[i][j].g = n;
        }
    }
}
void changeBlue(BMP* bitmap, int n){
    for(int i = 0; i<bitmap->bmih.height; i++){
        for(int j = 0; j<bitmap->bmih.width; j++){
            bitmap->arr[i][j].b = n;
        }
    }
```

```
}



void changeMostFrequentColour(BMP* bitmap, int r_new, int g_new, int
b_new){
    int*** most_frequent = calloc(256,sizeof(int**));
    {
        for (int i=0;i<256;i++){
            most_frequent[i]=calloc(256,sizeof(int*));
            for (int j=0;j<256;j++)
            {
                most_frequent[i][j]=calloc(256,sizeof(int));
            }
        }
    }
    for (int i=0;i<bitmap->bmih.height;i++){
        for (int j=0;j<bitmap->bmih.width;j++){
            most_frequent[bitmap->arr[i][j].r][bitmap-
>arr[i][j].g][bitmap->arr[i][j].b]++;
        }
    }
    int max = most_frequent[0][0][0];
    int max_r=0,max_g=0,max_b=0;
    for (int i=0;i<256;i++){
        for (int j=0;j<256;j++){
            for (int k=0;k<256;k++){
                if (most_frequent[i][j][k]>max)
                {
```

```
                    max_r=i;
                    max_g=j;
                    max_b=k;
                    max=most_frequent[i][j][k];
                }
            }
        }
    }
    for(int i = 0; i<bitmap->bmih.height; i++){
        for(int j = 0; j<bitmap->bmih.width; j++){
            if((bitmap->arr[i][j].r == max_r) && (bitmap->arr[i][j].g ==
max_g) && (bitmap->arr[i][j].b == max_b)){
                bitmap->arr[i][j].r = r_new;
                bitmap->arr[i][j].g = g_new;
                bitmap->arr[i][j].b = b_new;
            }


        }
    }
    for (int i=0;i<256;i++){
        for (int j=0;j<256;j++){
            free(most_frequent[i][j]);
        }
    }
    for (int i=0;i<256;i++){
        free(most_frequent[i]);
    }
```

```
    free(most_frequent);
}


void swapFour(BMP* bitmap, const char* path, int x1, int y1, int x2, int
y2, char option){
    BMP bmp_copy;
    readImage(path, &bmp_copy);
    int x, y;
    int fragment_width = (abs(x2 - x1))/2;
    int fragment_height = (abs(y1 - y2))/2;
    if (x1 > bitmap->bmih.width || y1 > bitmap->bmih.height || x2 > bit-
map->bmih.width || y2 > bitmap->bmih.height) {
        printf("Ошибка: координаты не должны быть за пределами
изображения.\n");
        return;
    }
    if(x1 < 0 || y1 < 0 || x2 < 0 || y2 < 0){
        printf("Ошибка: координаты не могут быть отрицательными.\n");
        return;
    }
    if (x1 > x2 || y2 > y1) {
        printf("Ошибка: координаты заданы неправильно.\n");
        return;
    }
```

```
    switch (option){
        case 'r':
            for (y = y2; y < y2 + fragment_height; y++){
                for (x = x1; x < x1 + fragment_width; x++){
                    bitmap->arr[y][x].r = bitmap->arr[y][x + frag-
ment_width].r;
                    bitmap->arr[y][x].g = bitmap->arr[y][x + frag-
ment_width].g;
                    bitmap->arr[y][x].b = bitmap->arr[y][x + frag-
ment_width].b;
                }
            }
            for (y = y2; y < y2 + fragment_height; y++) {
                for (x = x1 + fragment_width; x < x2; x++){
                    bitmap->arr[y][x].r = bitmap->arr[y + frag-
ment_height][x].r;
                    bitmap->arr[y][x].g = bitmap->arr[y + frag-
ment_height][x].g;
                    bitmap->arr[y][x].b = bitmap->arr[y + frag-
ment_height][x].b;
                }
            }
            for (y = y2 + fragment_height; y < y1; y++) {
                for (x = x1 + fragment_width; x < x2; x++) {
                    bitmap->arr[y][x].r = bitmap->arr[y][x - frag-
ment_width].r;
                    bitmap->arr[y][x].g = bitmap->arr[y][x - frag-
ment_width].g;
                    bitmap->arr[y][x].b = bitmap->arr[y][x - frag-
ment_width].b;
                }
```

```
            }
            for (y = y2 + fragment_height; y < y1; y++) {
                for (x = x1; x < x1 + fragment_width; x++) {
                    bitmap->arr[y][x].r =  bmp_copy.arr[y - frag-
ment_height][x].r;
                    bitmap->arr[y][x].g =  bmp_copy.arr[y - frag-
ment_height][x].g;
                    bitmap->arr[y][x].b =  bmp_copy.arr[y - frag-
ment_height][x].b;
                }
            }
            break;
        case 'd':
            for (y = y2 + fragment_height; y < y1; y++) {
                for (x = x1; x < x1 + fragment_width; x++) {
                    int t;
                    t = bitmap->arr[y][x].r;
                    bitmap->arr[y][x].r = bitmap->arr[y - frag-
ment_height][x + fragment_width].r;
                    bitmap->arr[y - fragment_height][x + frag-
ment_width].r = t;



                    t = bitmap->arr[y][x].g;
                    bitmap->arr[y][x].g = bitmap->arr[y - frag-
ment_height][x + fragment_width].g;
                    bitmap->arr[y - fragment_height][x + frag-
ment_width].g = t;



                    t = bitmap->arr[y][x].b;
```

```
                    bitmap->arr[y][x].b = bitmap->arr[y - frag-
ment_height][x + fragment_width].b;

                    bitmap->arr[y - fragment_height][x + frag-
ment_width].b = t;

                }

            }

            for (y = y2; y < y2 + fragment_height; y++) {

                for (x = x1; x < x1 + fragment_width; x++) {

                    int t;

                    t = bitmap->arr[y][x].r;

                    bitmap->arr[y][x].r = bitmap->arr[y + frag-
ment_height][x + fragment_width].r;

                    bitmap->arr[y + fragment_height][x + frag-
ment_width].r = t;



                    t = bitmap->arr[y][x].g;

                    bitmap->arr[y][x].g = bitmap->arr[y + frag-
ment_height][x + fragment_width].g;

                    bitmap->arr[y + fragment_height][x + frag-
ment_width].g = t;



                    t = bitmap->arr[y][x].b;

                    bitmap->arr[y][x].b = bitmap->arr[y + frag-
ment_height][x + fragment_width].b;

                    bitmap->arr[y + fragment_height][x + frag-
ment_width].b = t;

                }

            }

            break;
```

```
        default:
            printf("Опция должна принимать значения r или d\n");
            break;
    }
}
RGB changeColour(uint8_t r, uint8_t g, uint8_t b){
    RGB pixel;
    pixel.r = r;
    pixel.g = g;
    pixel.b = b;
    return pixel;
}


void drawpixel(int x,int y, BMP* bitmap, RGB colour){
    if(x < 0 || y < 0){
        printf("Ошибка: координаты не могут быть отрицательными.\n");
        return;
    }
    if (x > bitmap->bmih.width || y > bitmap->bmih.height) {
        printf("Ошибка: координаты не должны быть за пределами
изображения.\n");
        return;
    }
    bitmap->arr[y][x] = colour;
}
```

```
void drawLine(int x1, int y1, int x2, int y2, BMP* bitmap, RGB colour) {
    if (x1 > (bitmap->bmih.width) || y1 > (bitmap->bmih.height) || x2 > (bitmap->bmih.width) || y2 > (bitmap->bmih.height)){
        printf("Ошибка: координаты не должны быть за пределами изображения.\n");
        return;
    }
    if(x1 < 0 || y1 < 0 || x2 < 0 || y2 < 0){
        printf("Ошибка: координаты не могут быть отрицательными.\n");
        return;
    }
    const int deltaX = abs(x2 - x1);
    const int deltaY = abs(y2 - y1);
    const int signX = x1 < x2 ? 1 : -1;
    const int signY = y1 < y2 ? 1 : -1;
    int error = deltaX - deltaY;
    drawpixel(x2, y2, bitmap, colour);
    while(x1 != x2 || y1 != y2)
    {
        drawpixel(x1, y1, bitmap, colour);
        int error2 = error * 2;
        if(error2 > -deltaY)
        {
            error -= deltaY;
            x1 += signX;
        }
        if(error2 < deltaX)
        {
```

```
            error += deltaX;
            y1 += signY;
        }
    }
}


void drawSquare(BMP* bitmap, int x, int y, int side, int thickness, RGB
square_colour, int fill, RGB fill_colour){
    if(x < 0 || y < 0){
        printf("Ошибка: координаты не могут быть отрицательными.\n");
        return;
    }
    if(thickness < 1){
        printf("Ошибка: толщина не может быть меньше 1.\n");
        return;
    }
    if(side < 1){
        printf("Ошибка: сторона не может быть меньше 1.\n");
        return;
    }
    if (x > bitmap->bmih.width || y > bitmap->bmih.height) {
        printf("Ошибка: координаты не должны быть за пределами
изображения.\n");
        return;
    }
    if( (x + side > bitmap->bmih.width) || (y - side < 0)){
        printf("Ошибка: Сторона квадрата не должна выходить за пределы
изображения.\n");
```

```
        return;
    }
    if(x < thickness || (x + thickness) > bitmap->bmih.width || y <
thickness || (y + thickness) > bitmap->bmih.height){
        printf("Ошибка: Толщина изображения слишком велика.\n");
        return;
    }
    if(fill == 1){
        for(int i = y - side; i <= y; i++){
            for(int j = x; j < x + side; j++){
                bitmap->arr[i][j] = fill_colour;
            }
        }
    }
    x = x - thickness;
    y = y + thickness;
    for(int t = 0; t < thickness; t++){
        drawLine(x, y, x, y - side, bitmap, square_colour);
        x++;



    }
    for(int t = 0; t < thickness; t++){
        drawLine(x, y, x + side, y, bitmap, square_colour);
        y--;
    }
    for(int t = 0; t < thickness; t++){
        drawLine(x + side, y, x + side, y - side, bitmap, square_colour);
```

```
        x--;


    }
    for(int t = 0; t < thickness; t++){
        drawLine(x, y - side, x + side, y - side, bitmap, square_colour);
        y++;
    }
}



int checkPicture(const char* path){
    if(strlen(path) < 4){
        help();
        return 0;
    }
    if((path[strlen(path)-1] == 'p') && (path[strlen(path)-2] == 'm') &&
(path[strlen(path) - 3] == 'b') && (path[strlen(path) -4] == '.')){
        return 0;
    }
    printf("Файл не был передан\n");
    help();
    return 1;
}



void help(){
    printf("\n\tВас приветствует программа для редактирования bmp
файлов\n");
```

```
    printf("Поддерживаются только файлы BMP Version 3 с глубиной
изображения - 24 бита и без сжатия.\n");

    printf("Формат ввода: ./a.out -<короткий ключ>/--<длинный ключ>
<аргумент> ... <имя файла>.bmp\n\n");

    printf("Описание функций, которые выполняет программа. Пожалуйста,
передавайте аргументы через запятую!\n");

    printf("-s --square, рисование квадрата. Нужно указать такие
параметры, как:\n"

           "1)координаты верхнего левого угла\n"

           "2)размер стороны квадрата\n"

           "3)толщина линий\n"

           "4)цвет линий\n"

           "5)осуществлять заливку или нет (должен быть равен 1 для
осуществления заливки)\n"

           "6)если выбрана заливка, указать ее цвет\n"

           "ОБРАЗЕЦ: ./a.out -s/--square
<x>,<y>,<сторона>,<толщина>,<красный цвет квадрата>,\n<зеленый цвет
квадрата>,<синий цвет квадрата>,<опция заливки 1 или 0>,<красный цвет
заливки>,\n<зеленый цвет заливки>,<синий цвет заливки> <название
файла>.bmp\n\n");

    printf("-w --swap, поменять местами 4 куска области. Нужно указать
такие параметры, как:\n"

           "1)координаты верхнего левого угла области\n"

           "2)координаты правого нижнего угла области\n"

           "3)способ обмена частей: по кругу, по диагонали (r или d)\n"

           "ОБРАЗЕЦ: ./a.out -w/--swap <x1>,<y1>,<x2>,<y2><r или d>
<название файла>.bmp\n\n");

    printf("-c --changefreq, нахождение самого часто встречаемого цвета.
Нужно указать такие параметры, как:\n"

           "Указать, в какой цвет нужно перекрасить его.\n"

           "ОБРАЗЕЦ: ./a.out -c/--changefreq <красный>,<зеленый>,<синий>
<название файла>.bmp\n\n");
```

```
    printf("-f --filter, замена значения одного из цвета на другое
значение. Нужно указать такие параметры, как:\n"

           "1)какой из 3х цветов нужно заменить\n"

           "2)значение цвета\n"

           "ОБРАЗЕЦ: ./a.out -f/--filter <r/g/b>,<значение цвета (0-255)>
<название файла>.bmp\n\n");

    printf("-i --info, информация о файле.\n");

    printf("-h --help, инструкция к программе.\n");

}



int isNotSupported(BMP* bitmap){

    if(bitmap->bmfh.signature != 19778){

        printf("Неизвестный формат файла\n");

        return 1;

    }

    if(bitmap->bmih.compression != 0){

        printf("Неизвестный метод сжатия файла\n");

        return 1;

    }

    if(bitmap->bmih.colorsInColorTable != 0){

        printf("Версия файла не поддерживается\n");

        return 1;

    }

    return 0;

}
```