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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка изображений в формате BMP.

Студент гр. 1303 _____ Кузнецов Н.А.

Преподаватель _____ Чайка К.В.

Санкт-Петербург
2022

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Кузнецов Н.А.

Группа 1303

Тема работы работа с текстом в Си.

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 страниц.

Дата выдачи задания: 22.03.2022

Дата сдачи реферата: 19.05.2022

Дата защиты реферата: 21.05.2022

Студент _____ Кузнецов Н.А.

Преподаватель _____ Чайка. К.В.

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 Clion (JB).

1. ЗАДАНИЕ

Вариант 3.

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

1. 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
2. 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

3. 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
4. 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 РАБОТЫ

2.1. Структуры

1) *BitmapFileHeader* и *BitmapInfoHeader*

Содержат информацию о файле (версию *bmp*, ширину и высоту изображения и другое).

2) *Rgb*

Позволяет для каждого пикселя записать значение каждой из компонент.

3) *rgb_for_count*

Аналогичен *Rgb*, но также позволяет записать количество данного пикселя на изображении.

2.2. Функция фильтра *rgb*-компонент

1) *change_component*

С помощью цикла *for* у каждого пикселя заменяется значение нужной компоненты, на требуемое значения.

2) *print_square*

С помощью циклов *for* происходит закраска кромок рамки квадрата в нужный цвет, далее в зависимости от требования пользователя производится закраска внутренней области в нужный цвет

3) *change_four_ares*

Выбирается нужный прямоугольный участок на изображении и делится на 4 равные области, под каждую область создается свой массив *Rgb*, где записываются копии областей. В зависимости от требуемого типа перемещения данные массивы копируются в те области, в которые требуется.

4) *max_met_color*

Создается пустой массив *rgb_for_count*, для записи в него уникальных пикселей. Циклами *for* происходит перебор всех пикселей, находятся все уникальные пиксели и их количество. Далее находится самый чаще встречающийся цвет. Перебором всех пикселей избражения происходит

сравнение с нужным и в случае совпадения перекрашивается в нужный пользователю цвет.

5) *CLI*

Изначально создается *config*, куда будут записаны: факт вызова ключей отвечающих за вызов функций, значения переменных, используемые функциями и получаемые при помощи вспомогательных кючей. Далее с помощью цикла *while* вызывается функция *set_config,* в которую передается считанный ключ, и с использованием switch выбирается что нужно считать для данного ключа, эти значения передаются в *config*. При нехватке у какого либо из ключей данных происходит печать об ошибке и завершение программы.

6) Вызов функций

После заполнения *config* вызывается нужная функция и в нее передаются нужные данные. Если функциональных ключей несколько, то выводится информация об ошибке, функции не вызываются, а программа завершается. Так же здесь происходит проверка на корректность данных из *config*.

3. ЗАКЛЮЧЕНИЕ

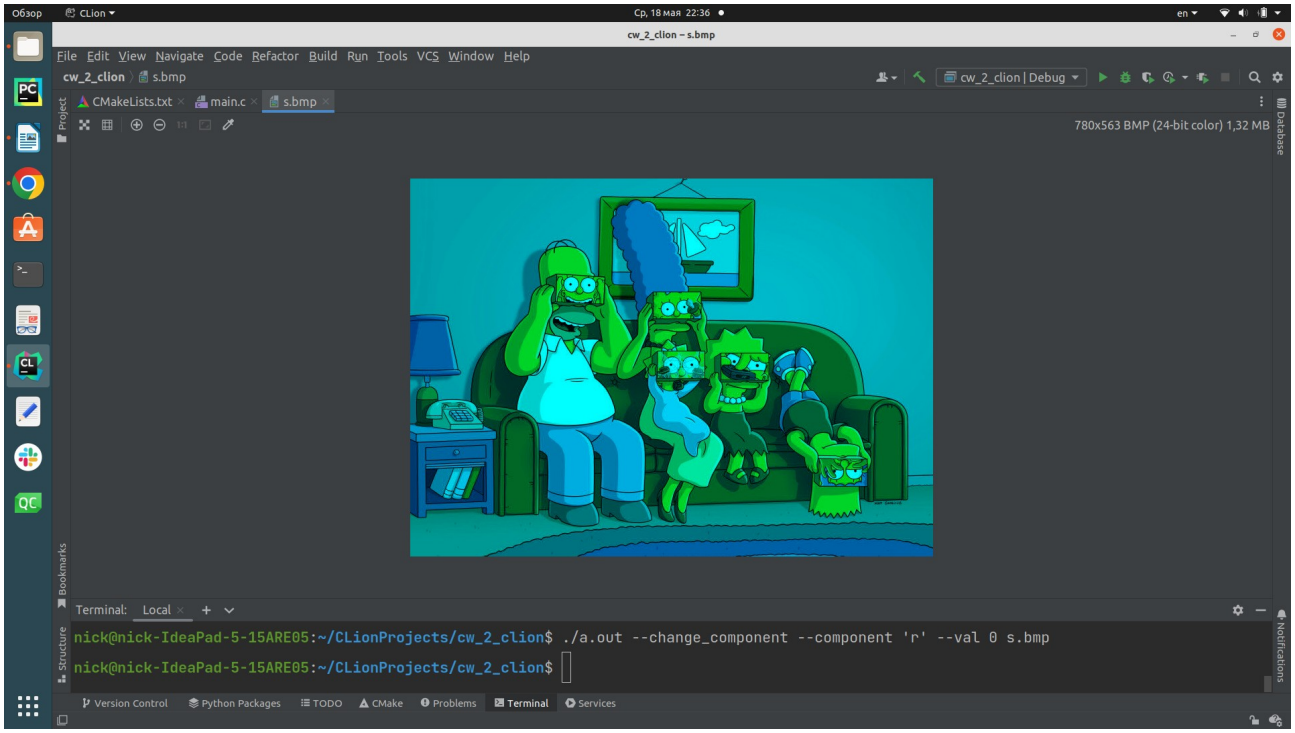
В результате выполнения курсовой работы было создано приложение с *CLI* для обработки изображений в формате BMP.

4. СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

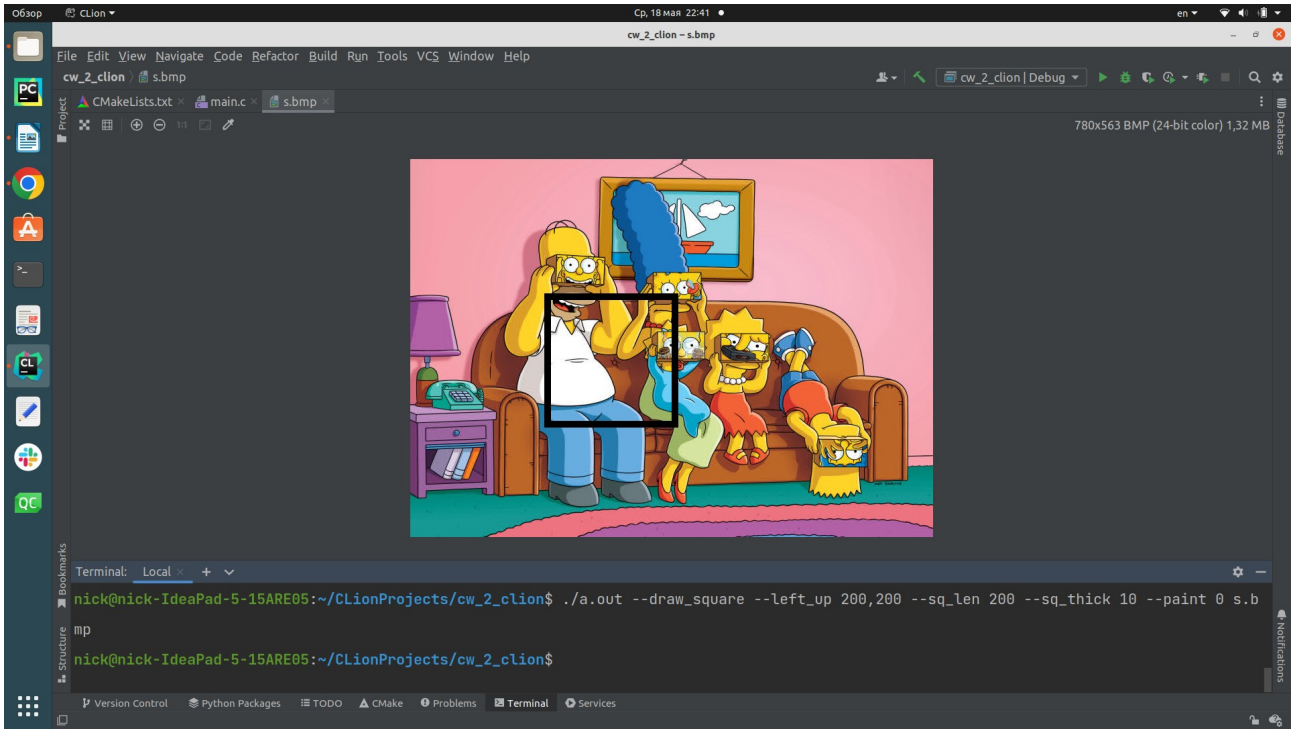
1. Керниган Б. И Ритчи Д. Язык программирования Си М.: Вильямс.
2. Информация о BMP файлах: https://ru.wikipedia.org/wiki/BMP

5. Приложение А. Тестирование

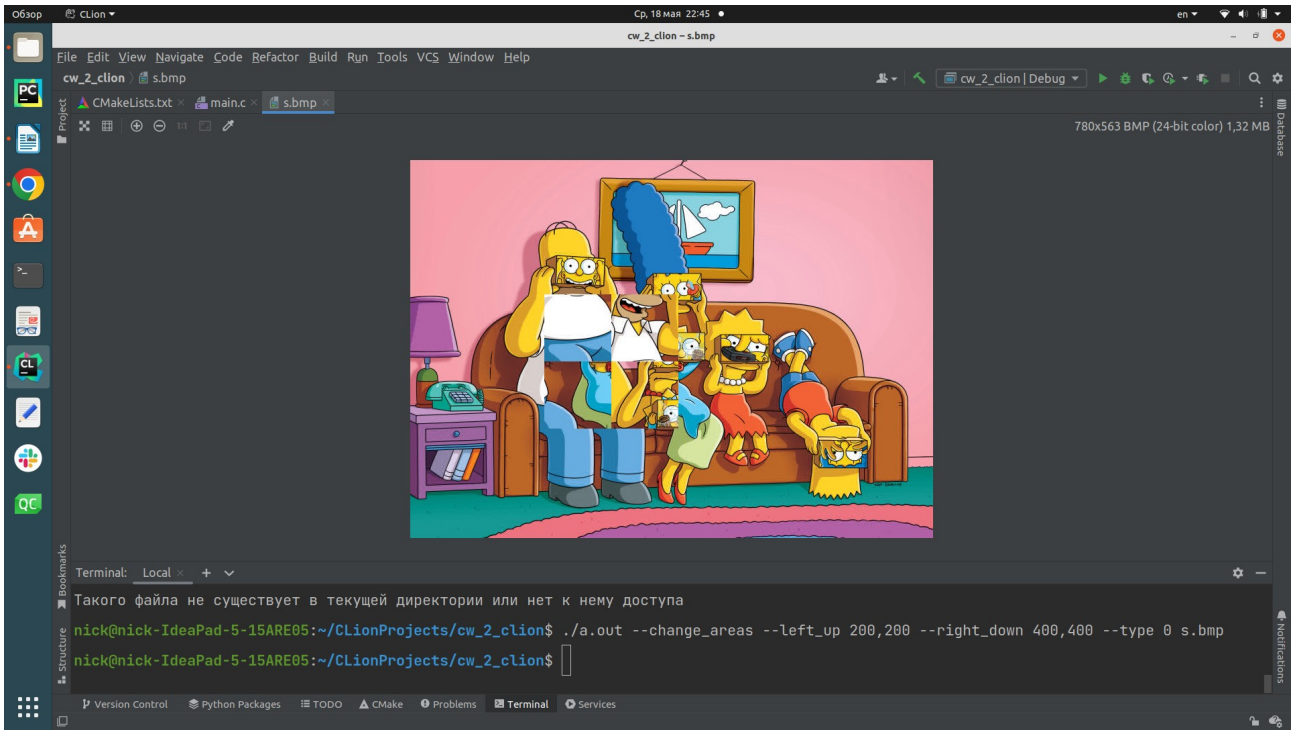
1) Фильтр *rgb*-компоненты



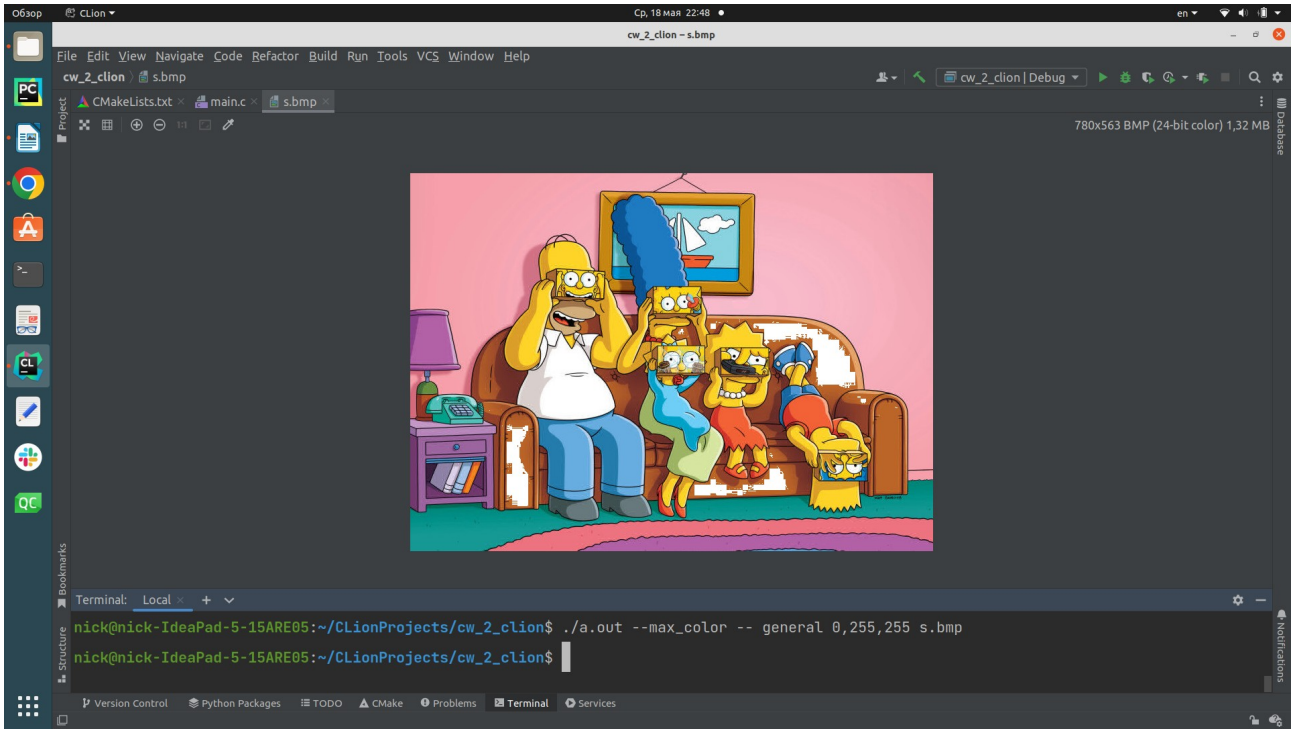
2) Рисование квадрата



3) Смена 4-ех областей



4) Нахождение чаще всего встречающегося цвета



6. Приложение Б. Исходный код программы

Файл cw.c

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

#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;

typedef struct
{
    unsigned char b;
    unsigned char g;
    unsigned char r;
    int kol;
} rgb_for_count;

typedef struct{
    // 1)
    char component; int val;
    // 2)
```

```
    int x0, y0, l_r, l_g, l_b, len, thick, paint, r, g, b;
    // 3)
    int x1, y1, type;
    // 4)
    // ----
    // func
    int help, info, ch_com, dr_sq, ch_areas, mx_co, clean;
} Config;

#pragma pack(pop)


void put_to_file(BitmapFileHeader* bmfh, BitmapInfoHeader* bmif, int W,
int H, Rgb** arr, char* file);


void printHelp(){
    printf("Руководство по использованию программы:\n");
    printf("-Программа обрабатывает BMP-файлы версии V3.\n");
    printf("-Для запуска программы необходимо передать следующие
аргументы:\n");
    printf("\t./a.out - имя исполняемого файла\n");
    printf("\t-f или --func - ключ для вызова функции (перечень указан в
списке ключей)\n");
    printf("\t-f или --func - ключи для для записи необходимых данных для
выполнения функций\n\t(перечень указан в списке ключей)\n");
    printf("\t<arg1>,<arg2>... - аргументы к ключам, если
требуются(указаны в списке ключей, аргументы разделяются запятой)\n");
    printf("\t<filename>  -- имя BMP-файла, который необходимо обработать.
Он должен находиться в текущей директории\n");
    printf("-Список функциональных ключей и их аргументы:\n");
    printf("\t--help или -H  (без аргументов) - вывод руководства по
использованию программы\n");
    printf("\t--info или -I  (без аргументов) - вывод информации о BMP-
файле, значения полей его заголовков\n");
    printf("\t--change_component или -c - замена значения одной из
компонент в каждом пикселе\n");
    printf("\t\tтребуются ключи: --component и --val\n");
    printf("\t--draw_square или -s - рисование квадарта с рамкой
произвольной длины, ширины и возможностью заливки произвольным цветом\n");
    printf("\t\tтребуются ключи: --left_up, --sq_len, --sq_thick, --
frame_rgb, --paint и --general_rgb\n");
    printf("\t--change_areas или -4 - выбор прямоугольного участка\n\tc
последующим его разбиением на 4 равные части и сменой этих частей местами\
n");
    printf("\t\tтребуются ключи: --left_up, --right_down и --type\n");
    printf("\t--max_color или -m - нахождение самого встречаемого цвета и
замена на новый\n");
    printf("\t\tтребуются ключи: general_rgb\n");
    printf("\t--clean или -C - очистка изображения\n");
    printf("-Список вспомогательных ключей\n");
    printf("\t--left_up или -u - принимает координаты левого верхнего
угла\n");
    printf("\t\tx0,y0\n");
    printf("\t--right_down или -d - принимает координаты правого нижнего
угла\n");
    printf("\t\tx1,y1\n");
    printf("\t--sq_len или -l - принимает длину квадрата\n");
```

```
    printf("\t--sq_thick или -t - принимает толщину рамки квадрата\n");
    printf("\t--paint или -p - если значение 1 - закрасить квадрат, 0 - не
закрашивать\n");
    printf("\t--frame_rgb или -f - цвет закраски рамки\n");
    printf("\t\tr,g,b - числа в диапопзоне [0;255]\n");
    printf("\t--general_rgb или -f - цвет пикселей\n");
    printf("\t\tr,g,b - числа в диапопзоне [0;255]\n");
    printf("\t--type или -y - тип перемещения областей: 1 - по диагонали,
0 - по кругу, по часовой\n");
    printf("\t--component или -o - какую компоненту нужно заменть
(значение 'r', 'g' или 'b'\n");
    printf("\t--val или -v - значение компонеты для замены\n");
    printf("\t\tr,g,b - число в диапопзоне [0;255]\n");
}

void printFileHeader(BitmapFileHeader header){
    printf("Информация о BMP-файле:\n");
    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 printInfoHeader(BitmapInfoHeader header){
    printf("\tПоля структуры BitmapInfoHeader:\n");
    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);
}



// Смена одной из компонент

void change_component(char component, int new_value, Rgb** arr, int W, int
H, BitmapFileHeader* bmfh, BitmapInfoHeader* bmif, char* file){
    if(component == 'r'){
        for(int i = 0; i < H; i++){
            for(int j = 0; j < W; j++){
                arr[i][j].r = new_value;
            }
```

```
        }
    } if (component == 'g'){
        for(int i = 0; i < H; i++) {
            for (int j = 0; j < W; j++) {
                arr[i][j].g = new_value;
            }
        }
    } if (component == 'b'){
        for(int i = 0; i < H; i++) {
            for (int j = 0; j < W; j++) {
                arr[i][j].b = new_value;
            }
        }
    }

    put_to_file(bmfh, bmif, W, H, arr, file);

}

//Рисовние квадрата

void print_square(int x0, int y0, int side_len, int thickness, int line_r,
int line_g, int line_b, int paint,
                  int square_r, int square_g, int square_b, int H, int W,
Rgb** arr, BitmapFileHeader* bmfh, BitmapInfoHeader* bmif, char* file){

    //Рисование верхней кромки

    for(int i = y0; i < y0 + thickness; i++){
        for(int j = x0; j < x0 + side_len; j++){
            arr[i][j].r = line_r;
            arr[i][j].g = line_g;
            arr[i][j].b = line_b;

        }
    }

    //Рисование нижней кромки

    for(int i = y0 + side_len - thickness; i < y0 + side_len; i++){
        for(int j = x0; j < x0 + side_len; j++){
            arr[i][j].r = line_r;
            arr[i][j].g = line_g;
            arr[i][j].b = line_b;

        }
    }

    //Рисование левой кромки

    for(int i = y0; i < y0 + side_len; i++){
        for(int j = x0; j < x0 + thickness; j++){
            arr[i][j].r = line_r;
            arr[i][j].g = line_g;
            arr[i][j].b = line_b;

        }
    }
```

```
    //Рисование правой кромки

    for(int i = y0; i < y0 + side_len; i++){
        for(int j = x0 + side_len - thickness; j < x0 + side_len; j++){
            arr[i][j].r = line_r;
            arr[i][j].g = line_g;
            arr[i][j].b = line_b;

        }
    }

    //Закрашивание квадрата

    if(paint == 1){
        for(int i = y0 + thickness; i < y0 + side_len - thickness; i++){
            for(int j = x0 + thickness; j < x0 + side_len - thickness; j +
+){
                arr[i][j].r = square_r;
                arr[i][j].g = square_g;
                arr[i][j].b = square_b;
            }
        }
    }

    put_to_file(bmfh, bmif, W, H, arr, file);

}

//4 области

void change_four_areas(int x0, int y0, int x1, int y1, int move_type,
Rgb** arr, int H, int W,
                       BitmapFileHeader* bmfh, BitmapInfoHeader* bmif,
char* file){
    int h = (y1 - y0)/2;
    int w = (x1 - x0)/2;

    //Выделение памяти для прямоугольников

    Rgb **arr_1 = malloc(H*sizeof(Rgb*));
    Rgb **arr_2 = malloc(H*sizeof(Rgb*));
    Rgb **arr_3 = malloc(H*sizeof(Rgb*));
    Rgb **arr_4 = malloc(H*sizeof(Rgb*));
    for(int i=0; i<h; i++){
        arr_1[i] = malloc(w * sizeof(Rgb));
        arr_2[i] = malloc(w * sizeof(Rgb));
        arr_3[i] = malloc(w * sizeof(Rgb));
        arr_4[i] = malloc(w * sizeof(Rgb));
    }

    //Заполнение прямооугольников

    for(int i = 0; i < h; i++){
        for(int j = 0; j < w; j++){
            arr_1[i][j] = arr[i + y0][j + x0];
            arr_2[i][j] = arr[i + y0][j + x0 + w];
            arr_3[i][j] = arr[i + y0 + h][j + x0];
```

```
            arr_4[i][j] = arr[i + y0 + h][j + x0 + w];
        }
    }

    //Перемещение по кругу

    if(move_type == 0){
        for(int i = 0; i < h; i++){
            for(int j = 0; j < w; j++){
                arr[i + y0][j + x0] = arr_3[i][j];
                arr[i + y0][j + x0 + w] = arr_1[i][j];
                arr[i + y0 + h][j + x0 + w] = arr_2[i][j];
                arr[i + y0 + h][j + x0] = arr_4[i][j];
            }
        }
    }

    //Перемещение по диагонали

    if(move_type == 1){
        for(int i = 0; i < h; i++){
            for(int j = 0; j < w; j++){
                arr[i + y0][j + x0] = arr_4[i][j];
                arr[i + y0 + h][j + x0 + w] = arr_1[i][j];
                arr[i + y0][j + x0 + w] = arr_3[i][j];
                arr[i + y0 + h][j + x0] = arr_2[i][j];
            }
        }
    }

    put_to_file(bmfh, bmif, W, H, arr, file);
    for(int i = 0; i < h; i++){
        free(arr_1[i]);
        free(arr_2[i]);
        free(arr_3[i]);
        free(arr_4[i]);
    }
    free(arr_1);
    free(arr_2);
    free(arr_3);
    free(arr_4);

}

//Самый часто встречаемый цвет

void max_met_color(int new_color_r, int new_color_g, int new_color_b,
Rgb** arr, int H, int W,
                   BitmapFileHeader* bmfh, BitmapInfoHeader* bmif, char*
file){
    rgb_for_count list[W*H];
    int color_kol = 0;
    for(int i = 0; i < H; i++){
        for(int j = 0; j < W; j++){
            //Выбрали пиксель
            int flag = 0;
            for(int k = 0; k < color_kol; k++){
```

```
                if(list[k].r == arr[i][j].r && list[k].g == arr[i][j].g &&
list[k].b == arr[i][j].b){
                    list[k].kol++;
                    flag = 1;
                    break;
                }
            }
            if(flag == 0){
                list[color_kol].r = arr[i][j].r;
                list[color_kol].g = arr[i][j].g;
                list[color_kol].b = arr[i][j].b;
                list[color_kol].kol = 1;
                color_kol++;
            }
        }
    }

    //Находим самый часто встречаемый элемент

    rgb_for_count max;
    max = list[0];
    for(int i = 1; i < color_kol; i++){
        if(list[i].kol > max.kol){
            max = list[i];
        }
    }

    //Заменяем его

    for(int i = 0; i < H; i++){
        for(int j = 0; j < W; j++){
            if(max.r == arr[i][j].r && max.g == arr[i][j].g && max.b ==
arr[i][j].b){
                arr[i][j].r = new_color_r;
                arr[i][j].g = new_color_g;
                arr[i][j].b = new_color_b;
            }
        }
    }

    put_to_file(bmfh, bmif, W, H, arr, file);

}

//Перевернуть изображение

void reverse(Rgb** arr, int H, int W){
    Rgb* help;
    for(int i = 0; i < H/2; i++){
        help = arr[i];
        arr[i] = arr[H - i - 1];
        arr[H - i - 1] = help;
    }
}

//Перезагрузка изоражения
```

```
void reload_picture(BitmapFileHeader* bmfh, BitmapInfoHeader* bmif, char*
file){
    FILE* f = fopen("s2.bmp", "rb");
    BitmapFileHeader bmfh2;
    BitmapInfoHeader bmif2;
    fread(&bmfh2,1,sizeof(BitmapFileHeader),f);
    fread(&bmif2,1,sizeof(BitmapInfoHeader),f);
    unsigned int H_2 = bmif2.height;
    unsigned int W_2 = bmif2.width;
    Rgb **arr = malloc(H_2 * sizeof(Rgb*));
    unsigned int offset = (W_2 * sizeof(Rgb)) % 4;
    offset = (offset ? 4-offset : 0);
    for(int i=0; i<H_2; i++){
        arr[i] = malloc(W_2 * sizeof(Rgb) + offset);
        fread(arr[i], 1, W_2 * sizeof(Rgb) + offset, f);
    }

    bmif->width = W_2;
    bmif->height = H_2;

    reverse(arr, H_2, W_2);

    put_to_file(bmfh, bmif, W_2, H_2, arr, file);


}

//Вывод изображения

void put_to_file(BitmapFileHeader* bmfh, BitmapInfoHeader* bmif, int W,
int H, Rgb** arr, char* file){
    FILE *ff = fopen(file, "wb");

    bmif->height = H;
    bmif->width = W;

    int offset = (W * sizeof(Rgb)) % 4;
    offset = (offset ? 4-offset : 0);

    fwrite(bmfh, 1, sizeof(BitmapFileHeader),ff);
    fwrite(bmif, 1, sizeof(BitmapInfoHeader),ff);
    unsigned int w = W * sizeof(Rgb) + offset;

    reverse(arr, H, W);

    for(int i=0; i<H; i++){
        fwrite(arr[i],1,w,ff);
        free(arr[i]);
    }
    free(arr);
    fclose(ff);
}

int check_color(int r, int g, int b){
    if(r < 0 || r > 255 || g < 0 || g > 255 || b < 0 || b > 255){
        return 1;
    }
    return 0;
```

```
}

int set_config(Config* config, int opt){
    int count;
    switch (opt) {
        // func
        case 'H':
            config->help = 1;
            break;
        case 'I':
            config->info = 1;
            break;
        case 'c':
            config->ch_com = 1;
            break;
        case 's':
            config->dr_sq = 1;
            break;
        case '4':
            config->ch_areas = 1;
            break;
        case 'm':
            config->mx_co = 1;
            break;
        case 'C':
            config->clean = 1;
            break;
        // other keys
        case 'u':
            count = sscanf(optarg, "%d,%d", &config->x0, &config->y0);
            if(count != 2){
                return 1;
            }
            break;
        case 'd':
            count = sscanf(optarg, "%d,%d", &config->x1, &config->y1);
            if(count != 2){
                return 1;
            }
            break;
        case 'l':
            count = sscanf(optarg, "%d", &config->len);
            if(count != 1){
                return 1;
            }
            break;
        case 't':
            count = sscanf(optarg, "%d", &config->thick);
            if(count != 1){
                return 1;
            }
            break;
        case 'p':
            count = sscanf(optarg, "%d", &config->paint);
            if(count != 1){
                return 1;
            }
            break;
```

```
        case 'g':
            count = sscanf(optarg, "%d,%d,%d", &config->r, &config->g,
&config->b);
            if(count != 3){
                return 1;
            }
            break;
        case 'f':
            count = sscanf(optarg, "%d,%d,%d", &config->l_r, &config->l_g,
&config->l_b);
            if(count != 3){
                return 1;
            }
            break;
        case 'y':
            count = sscanf(optarg, "%d", &config->type);
            if(count != 1){
                return 1;
            }
            break;
        case 'o':
            count = sscanf(optarg, "%c", &config->component);
            if(count != 1){
                return 1;
            }
            break;
        case 'v':
            count = sscanf(optarg, "%d", &config->val);
            if(count != 1){
                return 1;
            }
            break;
    }
    return 0;
}

int main(int argc, char* argv[]){
    char* file = malloc(100*sizeof(char));
    strcpy(file, argv[argc - 1]);
    FILE *f = fopen(file, "rb");
    if (!f){
        printf("Такого файла не существует в текущей директории или нет к
нему доступа\n");
        return 0;
    }
    BitmapFileHeader bmfh;
    BitmapInfoHeader bmif;

    fread(&bmfh,1,sizeof(BitmapFileHeader),f);
    if(bmfh.signature != 0x4d42){
        printf("Файл не соответствует формату BMF\n");
        return 0;
    }

    fread(&bmif,1,sizeof(BitmapInfoHeader),f);
    if(bmif.bitsPerPixel != 24){
        printf("Изображение не содержит 24 бита на цвет\n");
        return 0;
```

```
}

if(bmif.headerSize != 40){
    printf("Данная версия BMP файла не обрабатывается\n");
    return 0;
}

unsigned int H = bmif.height;
unsigned int W = bmif.width;

Rgb **arr = malloc(H * sizeof(Rgb*));

int offset = (W * sizeof(Rgb)) % 4;
offset = (offset ? 4-offset : 0);

for(int i=0; i<H; i++){
    arr[i] = malloc(W * sizeof(Rgb) + offset);
    fread(arr[i], 1, W * sizeof(Rgb) + offset, f);
}

//Перевернутое изображение


reverse(arr, H, W);

struct option Opts[] = {
        {"help", no_argument, NULL, 'H'},
        {"info", no_argument, NULL, 'I'},
        {"change_component", no_argument, NULL, 'c'},
        {"draw_square",  no_argument, NULL, 's'},
        {"change_areas",  no_argument, NULL, '4'},
        {"max_color",  no_argument, NULL, 'm'},
        {"clean", no_argument, NULL, 'C'},
        {"left_up", required_argument, NULL, 'u'},
        {"right_down", required_argument, NULL, 'd'},
        {"sq_len", required_argument, NULL, 'l'},
        {"sq_thick", required_argument, NULL, 't'},
        {"paint", required_argument, NULL, 'p'},
        {"frame_rgb", required_argument, NULL, 'f'},
        {"general_rgb", required_argument, NULL, 'g'},
        {"type", required_argument, NULL, 'y'},
        {"component", required_argument, NULL, 'o'},
        {"val", required_argument, NULL, 'v'}
};

Config config = {'r', 0, 0, 0,
                 0,0, 0, 100, 20, 0,
                 255, 255, 255, 200, 200, 0,
                 0, 0, 0, 0, 0, 0, 0};

char* opts = "HIcs4mCu:d:l:t:p:f:g:y:";

int opt, longIndex;

opt = getopt_long(argc, argv, opts, Opts, &longIndex);

while(opt != -1){
    if(set_config(&config, opt) == 1){
```

```
            printf("Неверное количество аргументов у одного или нескольких
ключей\n");
            fclose(f);
            return 0;
        }
        opt = getopt_long(argc, argv, opts, Opts, &longIndex);
    }

    if(config.info + config.help + config.ch_com + config.ch_areas +
config.mx_co + config.dr_sq + config.clean == 0){
        printf("Не введены функцианальные ключи\n");
    } else if(config.info + config.help + config.ch_com + config.ch_areas
+ config.mx_co + config.dr_sq + config.clean > 1){
        config.info = 0; config.help = 0; config.ch_com = 0;
config.ch_areas = 0; config.mx_co = 0;
        config.dr_sq = 0; config.clean = 0;
        printf("Функцианальный ключ должен быть один\n");
    }

    // Функциональные ключи помощи

    if(config.help == 1){
        printHelp();
    }

    if(config.info == 1){
        printFileHeader(bmfh);
        printInfoHeader(bmif);
    }

    if(config.clean == 1){
        reload_picture(&bmfh, &bmif, file);
    }

    // Функциональные ключи действий
    if(config.ch_areas == 1){
        if(config.x0 < 0 || config.x1 < 0 || config.y0 < 0 || config.y1 <
0){
            printf("Координаты не могут быть отрицательными\n");
        } else if (config.x0 > bmif.width || config.x1 > bmif.width  ||
config.y0 > bmif.height || config.y1 > bmif.height ){
            printf("Одна или несколько координат имеют значения больше
размеров изображения\n");
        } else if(config.x0 > config.x1 || config.y0 > config.y1){
            printf("Координаты верхнего левого угла должны быть меньше
координиат правого нижнего\n");
        } else if(config.x1 - config.x0 < 2 || config.y1 - config.y0 < 2){
            printf("В одном пикселе невозможно выделить 4 области\n");
        } else if(config.type != 1 && config.type != 0){
            printf("Неверное значение типа перемещения\n");
        } else{
            change_four_areas(config.x0, config.y0, config.x1, config.y1,
config.type, arr, H, W, &bmfh, &bmif, file);
        }
    }

    if(config.ch_com == 1){
```

```
        if(config.component != 'r' && config.component != 'g' &&
config.component != 'b'){
            printf("Неверное название компоненты цвета\n");
        } else if(config.val < 0 || config.val > 255){
            printf("Неверное значение компоненты цвета\n");
        } else{
            change_component(config.component, config.val, arr, W, H,
&bmfh, &bmif, file);
        }
    }

    if(config.mx_co == 1){
        if(check_color(config.r, config.g, config.b)){
            printf("Значение одной или нескольких компопнент цвета
неверно\n");
        } else{
            max_met_color(config.r, config.g, config.b, arr, H, W, &bmfh,
&bmif, file);
        }
    }

    if(config.dr_sq == 1){
        if(config.x0 < 0 || config.x0 > bmif.width - 3 || config.y0  < 0
|| config.y0 > bmif.height - 3){
            printf("Координаты левого верхнего угла квадрата лежат вне
изображения\n");
        } else if(config.len < 3){
            printf("Длина стороны квадрата не может быть меньше 3\n");
        } else if(config.x0 + config.len > bmif.width || config.y0 +
config.len > bmif.height){
            printf("Квадрат не помещается в изображении\n");
        } else if(config.thick < 1){
            printf("Ширина рамки не может быть меньше 1\n");
        } else if(config.thick > config.len / 2 - 1){
            printf("Ширина рамки слишком большая\n");
        } else if(check_color(config.l_r, config.l_g, config.l_b)){
            printf("Значение одной или нескольких компопнент цвета рамки
неверно\n");
        } else if(config.paint != 0 && config.paint != 1){
            printf("Значение требования заливки неверно\n");
        } else if(check_color(config.r, config.g, config.b)){
            printf("Значение одной или нескольких компопнент цвета заливки
неверно\n");
        } else {
            print_square(config.x0, config.y0, config.len, config.thick,
config.l_r,
                         config.l_g, config.l_b, config.paint,
                         config.r, config.g, config.b, H, W, arr, &bmfh,
&bmif, file);
        }
    }

    if(config.help == 1 || config.info == 1){
        for(int i = 0; i < H; i++){
            free(arr[i]);
        }
        free(arr);
    }
```

```
    fclose(f);

    return 0;
}
```