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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка изображений

Студент гр. 3344 _____ Гусева Е.А.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2024

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Гусева Е.А.

Группа 3344

Тема работы: Обработка изображений.

Исходные данные:

- Программу требуется реализовать в виде утилиты, подобной стандартным *linux*-утилитам.
- Программа должна считать *bmp*-файл без сжатия с 24 битами на цвет
- Программа должна сохранить обработанный *bmp*-файл
- Все поля стандартных *BMP* заголовков в выходном файле должны иметь те же значения что и во входном
- Необходимо использовать *Makefile* для сборки проекта, название исполняемого файла должно быть: *cw*.

Содержание пояснительной записки:

- Содержание
- Введение
- Описание варианта работы
- Описание функций программы
- Описание структуры файлов программы
- Описание сборки проекта
- Примеры работы программы
- Заключение
- Список использованных источников

Предполагаемый объем пояснительной записки:

Не менее 45 страниц.

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 22.05.2024

Дата защиты реферата: 22.05.2024

Студент _____ Гусева Е.А.

Преподаватель _____ Глазунов С.А.

АННОТАЦИЯ

Данная курсовая посвящена разработке программы на языке C, она основана на структурах *BMPFile* и *rgb*. Программа предоставляет набор инструментов для рисования и преобразования изображений. Рисование включает в себя создание прямоугольников и шестиугольников с заданными параметрами и заливку их цветом. Преобразование включает в себя выбор и копирования заданной области и последующая вставку в указанное место картинки . Операции выполняются с использованием динамической памяти. Основное внимание уделено проверке корректности входных файлов, поддержке различных версий формата BMP, а также правильному выравниванию данных для записи входного файла. Для управления функциональностью программы используется интерфейс командной строки CLI. Проект структурирован на отдельные функции, сгруппированные в несколько файлов и поддерживает сборку с Make(CMake). Результатом работы программы является обработанное изображение с учетом выполненных операций.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной работы является создание программы на языке программирования C, которая будет обрабатывать BMP-изображение с помощью CLI интерфейса.

Для достижения поставленной цели требуется решить следующие задачи:

1. Изучить формат BMP
2. Изучить методы реализации CLI интерфейса
3. Реализовать функций обработки изображения
4. Реализовать эффективную сборку программы
5. Предусмотреть возможные ошибки и их причины

Возможные методы решения поставленных задач:

1. Реализация функций считывания и записи bmp-файлов
2. Реализация структур-хедеров для считанного изображения
3. Использование библиотеки getopt для работы с командной строкой
4. Сборка проекта с помощью Makefile
5. Вынесение каждой подзадачи в отдельную функцию

1. ОПИСАНИЕ ВАРИАНТА РАБОТЫ

Программа должна иметь следующую функции по обработке изображений:

- (1) Рисование прямоугольника. Флаг для выполнения данной операции: `--rect`. Он определяется:
  - Координатами левого верхнего угла. Флаг `--left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y
  - Координатами правого нижнего угла. Флаг `--right_down`, значение задаётся в формате `right.down`, где right – координата по x, down – координата по y
  - 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)
  - Прямоугольник 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) Рисование правильного шестиугольника. Флаг для выполнения данной операции: `--hexagon`. Шестиугольник определяется:
  - координатами его центра и радиусом в который он вписан. Флаги `--center` и `--radius`. Значение флаг `--center` задаётся в формате `x.y`, где x – координата по оси x, y – координата по оси y. Флаг `--radius` 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)
  - шестиугольник может быть залит или нет. Флаг `--fill`. Работает как бинарное значение: флага нет – false , флаг есть – true.
  - цветом которым залит шестиугольник, если пользователем выбран залитый. Флаг `--fill_color` (работает аналогично флагу `--color`)
- (3) Копирование заданной области. Флаг для выполнения данной операции: `--copy`. Функционал определяется:
  - Координатами левого верхнего угла области-источника. Флаг `--left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y

- Координатами правого нижнего угла области-источника. Флаг `--right_down`, значение задаётся в формате `right.down`, где right – координата по x, down – координата по y
- Координатами левого верхнего угла области-назначения. Флаг `--dest_left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y

Все подзадачи, ввод/вывод должны быть реализованы в виде отдельной функции.

2. ОПИСАНИЕ ПРОГРАММЫ

2.1. Описание функций программы

В программе используется класс *BMPFile*, которая представляет изображение и включает в себя заголовок файла *BitmapFileHeader*, заголовок информации *BitmapInfoHeader*, а также указатель на массив пикселей data. Также используется структура *rgb*, которая представляет из себя 3 числа *r, g, b,* отвечающих за 3 канала пикселя.

Функции и их краткое описание:

- BMPFile* openBMPFile(char* filename)
  Открывает файл BMP по указанному имени и загружает его заголовки
- void readRowsBMPFile(FILE* filepath, BMPFile* img)
  Считывает пиксельные данные файла в структуру
- BMPFile* readBMPFile(char* filename)
  Читает файл и возвращает указатель на структуру, содержащую заголовки и данные изображения
- void writeBMPFile(char* filename, BMPFile* img)
  записывает данные из структуру в файл с указанным именем
- void freeBMPFile(BMPFile* img) \
  освобождает память
- int is_bmp(char* arg)
  проверка что файл имеет нужное расширение
- void draw_pixel(BMPFile *img, int x, int y, rgb color)
  устанавливает цвет пикселя
- void fill_circle(BMPFile *img, int x0, int y0, int r, rgb color)
  рисует закрашенный круг на изображении

- void drawingLine(coords start, coords end, rgb color, unsigned int thickness, BMPFile* img)
  рисует линию
- void fill_zone(int i, int j, rgb color, rgb fill_color, BMPFile* img)
  функция по рекурсивной заливке фигуры
- int isInsideArea(coords polygonPoints[], int nAngle, coords point)
- int isInsideArea1(coords polygonPoints[], int nAngle, coords point)
  эти две функции проверяют находится ли точка внутри определенной зоне, без единицы верхняя половина шестиугольника, с единицей в нижней половине
- void swap(int* a, int* b)
  функция меняет значения местами
- void drawingRectangle(rectangle data, BMPFile* img)
  рисование прямоугольника
- void drawingHexagon(hexagon data, BMPFile* img)
  рисование шестиугольника
- void copy_and_paste_area(BMPFile* img, coords start, coords end, coords dest_start)
  копирование и вставление определенной области
- void coordschecker(char * coords)
  проверка координат
- void distancechecker(char * distance)
  проверяет корректность числа
- int *parse_color(char *color)
  разделяет строку, содержащую три компоненты цвета, разделенные точками, и возвращает массив из трех целых чисел
- void colorchecker(char *color)
  проверяет корректность строки цвета

- void thickNradiuschecker(char *thickness)
  функция для проверки толщины и радиуса
- void nameschecker(char* inputname, char* outputname)
  проверяет, являются ли имена входного и выходного файлов одинаковыми
- void bmp_namechecker(char * name)
  проверка корректности имени файла
- void count_argscheck(char * first,char * second,char * third, char * name)
  проверяет количество аргументов командной стороки для определенный команды
- void count_argscheck(char * first,char * second,char * third, char * name)
  проверяет отсутствие аргументов для определенной команды
- void print_file_header(BitmapFileHeader header)
  печатает информацию
- void print_info_header(BitmapInfoHeader header)
  печатает информацию
- int main(int argc, char *argv[])
  Основная функция программы, которая считывает опции командной строки, создает объект изображения, выполняет необходимые операции и записывает изображение в файл.

2.2. Описание структуры файлов программы

В программе предусмотрены следующие файлы:

- *Makefile*: Файл для автоматизации процесса компиляции и сборки программы.
- *work_with_bmp.c* файл для работы с файлами формата BMP
- *work_with_bmp.h* заголовочный файл

- functions.c файл со всеми функциями исполняющими задания для курсовой
- functions.h заголовоный файл для файла с функциями по заданиям
- checkers.c файл где содержатся функции для проверки корректности ввода данных
- checkers.h заголовочный файл для функций по проверки корректности ввода данных
- structs.h заголовочный файл с основными структурами
- main.c файл с основными функциями и с вспомогательной структурой

Эта структура файлов позволяет организовать код программы в логически связанные блоки, что облегчает его понимание и поддержку.

2.3. Описание сборки проекта

Для сборки проекта используется Makefile:

- cw – исполняемый файл, требует все нижеперечисленные объектные файлы для сборки и линковки.
- main.o - объектный файл, требующий main.c structs.h functions.h work_with_bmp.h checkers.h для компиляции.
- work_with_bmp.o - объектный файл, требующий work_with_bmp.c structs.h work_with_bmp.h для компиляции.
- functions.o - объектный файл, требующий functions.c structs.h functions.h для компиляции.
- checkers.o - объектный файл, требующий checkers.c, structurs.h, checkers.h для компиляции.
- Clean - очистка всех объектных файлов и исполняемого файла cw.

Компиляция происходит с помощью: gcc.

3. ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

Пример 1: Рисование квадрата.

Ввод	Вывод
./cw –rect –left_up 10.10 –right_down 100.100 –thickness 2 –color 255.0.0 –fill –fill_color 0.0.255 Airplane.bmp	Правильно измененная картинка

Пример 2: Рисование шестиугольника.

Ввод	Вывод
./cw –hexagon –center 150.150 –radius 100 –thickness 5 –color 255.0.0 –fill –fill_color 0.0.255 Airplane.bmp	Правильно измененная картинка

Пример 3: Копирование заданной области.

Ввод	Вывод
--copy –left_up 100.100 –right_down 150.150 –dest_left_up 200.200 Airplane.bmp	Правильно измененная картинка

Пример 4: Вывод информации об изображении.

Ввод	Вывод
-I input.bmp	

Пример 6: Проверка обработки ошибок.

Ввод	Вывод
-r --component_name green --component_value 100.100.100 input.bmp	Course work for option 4.9, created by Elena Guseva Error in parsing arguments: --

	component_value argument is incorrect

ЗАКЛЮЧЕНИЕ

Была успешно создана программа, которая обрабатывает изображение в зависимости от подаваемых пользователем флагов и аргументов в командную строку. Программа выполняет поставленные задачи по считыванию, обработке и записи BMP-изображений. При выполнении задания были улучшены навыки работы с изображением, также был получен опыт использование CLI интерфейса, реализованного при помощи библиотеки getopt. Полученные результаты показывают, что поставленные цели были успешно достигнуты.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Cplusplus. URL: https://cplusplus.com/reference/ (Дата обращения 18.05.2024)
2. Базовые сведения к выполнению курсовой и лабораторных работ по дисциплине «программирование». Второй семестр: учеб.-метод. пособие др. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2024. 36 с.
3. Geeksforgeeks. URL: https://www.geeksforgeeks.org (Дата обращения 20.05.2024)

ПРИЛОЖЕНИЕ А
КОД ПРОГРАММЫ

main.c

```
#include "structs.h"
#include "work_with_bmp.h"
#include "functions.h"
#include "checkers.h"

void print_file_header(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:s\t%x (%hu)\n", header.reserved2, header.reserved2);
    printf("pixelArrOffset:\t%x (%u)\n", header.pixelArrOffset,
header.pixelArrOffset);
}

void print_info_header(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);
    }


    struct {
        int rect,
        hexagon,
        copy,
        info,
        input,
        output,
        help,
        center,
        fill,
        collage,
        thickness;
    //!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    } flags;

    typedef struct copy{
        coords start, end, dest_start;
    }copy;

    typedef struct collage{
        coords start, end;
        int n,m;
    }collage;
    static struct option long_options[] = {
        {"help", no_argument, 0, 'h'},
        {"info", no_argument, 0, 'I'},
        {"input", required_argument, 0, 'i'},
        {"output", required_argument, 0, 'o'},
```

```
        {"rect", no_argument, 0, 'p'},
        {"left_up", required_argument, 0, 'u'},
        {"right_down", required_argument, 0, 'd'},
        {"thickness", required_argument, 0, 't'},
        {"color", required_argument, 0, 'c'},
        {"fill", no_argument, 0, 'f'},
        {"fill_color", required_argument, 0, 'z'},

        {"hexagon", no_argument, 0, 'x'},
        {"center", required_argument, 0, 'e'},
        {"radius", required_argument, 0, 'r'},

        {"copy", no_argument, 0, 'k'},
        {"dest_left_up", required_argument, 0, 's'},
        {"collage", no_argument, 0, 'q'},
        {"n", required_argument, 0, 'n'},
        {"m", required_argument, 0, 'm'},
        {NULL, 0, NULL, 0}
    };
    //BMPFile* img = readBMPFile("Airplane.bmp");hell-guss@hellguss-
HP:~/Рабочий стол/cw_2$ gcc main.c                    ^
    //writeBMPFile("fill.bmp", img);
    int main(int argc, char** argv)
    {

        BMPFile* img = NULL;

        rectangle rectangle = {0};
        hexagon hexagon = {0};
        copy cope = {0};
        collage col;
        int n = 0;
        int m = 0;
```

```
        coords left_up;
        coords rigth_down;
        rgb color;

        int fill = 0;
        rgb fill_color;
        int thickness;

        flags.info = 0;
        flags.fill = 0;

        char *input_name, *output_name;
        char *short_options="kpxhIi:u:d:t:c:f:e:r:s:o:z:q:";
        int opt, option_index = 0;
        opterr = 0;
        while ((opt=getopt_long_only(argc, argv, short_options, long_options,
&option_index)) != -1){
            //printf("%c\n", opt);
            //printf("%d\n", option_index);
            switch (opt){
                case 'i':
                    bmp_namechecker(optarg);
                    flags.input = 1;
                    input_name = optarg;
                    break;
                case 'o':
                    flags.output = 1;
                    output_name = optarg;
                    break;
                case 'h':
                    no_argschecker(argv[optind], "--help");
                    flags.help = 1;
                    break;
                case 'I':
```

```
                no_argschecker(argv[optind], "--info");
                flags.info = 1;
                break;
            case 'p':
                no_argschecker(argv[optind], "--rect");
                flags.rect = 1;
                break;
            case 'u':
                count_argscheck(argv[optind - 1], argv[optind], argv[optind + 1], "--
left_up");
                coordschecker(optarg);
                sscanf(optarg, "%d.%d", &left_up.x, &left_up.y);
                break;
            case 'd':
                count_argscheck(argv[optind - 1], argv[optind], argv[optind + 1], "--
right_down");
                coordschecker(optarg);
                sscanf(optarg, "%d.%d", &rigth_down.x, &rigth_down.y);
                break;
            case 't':
                count_argscheck(argv[optind - 1], argv[optind], argv[optind + 1], "--
thickness");
                thickchecker(optarg);
                sscanf(optarg, "%d", &thickness);
                //printf("%d", thickness);
                break;
            case 'c':
                count_argscheck(argv[optind - 1], argv[optind], argv[optind + 1], "--
color");
                colorchecker(optarg);
                sscanf(optarg, "%hhd.%hhd.%hhd", &color.red, &color.green,
&color.blue);
                break;
            case 'f':
```

```
                flags.fill = 1;
                fill = 1;
                break;
            case 'z':
                count_argscheck(argv[optind - 1], argv[optind], argv[optind + 1], "--
fill_color");
                colorchecker(optarg);
                sscanf(optarg, "%hhd.%hhd.%hhd", &fill_color.red,
&fill_color.green, &fill_color.blue);
                break;
            case 'x':
                no_argschecker(argv[optind], "--hexagon");
                flags.hexagon = 1;
                break;
            case 'e':
                count_argscheck(argv[optind - 1], argv[optind], argv[optind + 1], "--
center");
                coordschecker(optarg);
                flags.center = 1;
                sscanf(optarg, "%d.%d", &hexagon.centre.x, &hexagon.centre.y);
                break;
            case 'r':
                count_argscheck(argv[optind - 1], argv[optind], argv[optind + 1], "--
radius");
                thickchecker(optarg);
                sscanf(optarg, "%d", &hexagon.radius);
                break;
            case 'k':
                no_argschecker(argv[optind], "--copy ");
                flags.copy = 1;
                break;
            case 's':
                count_argscheck(argv[optind - 1], argv[optind], argv[optind + 1], "--
dest_left_up");
```

```
                coordschecker(optarg);
                sscanf(optarg, "%d.%d", &cope.dest_start.x, &cope.dest_start.y);
                break;
            case 'q':
                no_argschecker(argv[optind], "--collage ");
                flags.collage = 1;
                break;
            case 'n':
                count_argscheck(argv[optind - 1], argv[optind], argv[optind + 1], "--
n");
                sscanf(optarg, "%d", &n);
                break;
            case 'm':
                count_argscheck(argv[optind - 1], argv[optind], argv[optind + 1], "--
m");
                sscanf(optarg, "%d", &m);
                break;
            default:
                printf("fuck u");
        }

    }

    if(flags.help){
        printf("Course work for option 4.9, created by Elena Guseva.\n");
        if(!(flags.rect || flags.hexagon || flags.copy)){
            return 0;
        }
    }

    if(!flags.input){
        bmp_namechecker(argv[argc - 1]);
        input_name = argv[argc - 1];
    }
```

```
if(!flags.output){
    output_name = "out.bmp";
}

nameschecker(input_name, output_name);
img = readBMPFile(input_name);

if(flags.info == 1){
    if(input_name){
    print_file_header(img->bmfh);
    print_info_header(img->bmih);
    } else {
    printf("Error: BMPii file not loaded.\n");
    }
}


if(flags.rect){
    rectangle.start = left_up;
    rectangle.end = rigth_down;
    rectangle.thickness = thickness;
    rectangle.color = color;
    rectangle.fill_color = fill_color;
    rectangle.filled = fill;
    drawingRectangle(rectangle, img);
}

if(flags.hexagon){
    hexagon.color = color;
    hexagon.fill_color = fill_color;
    hexagon.filled = fill;
    hexagon.thickness = thickness;
    drawingHexagon(hexagon, img);
```

```
    }

    if(flags.copy){
        cope.start = left_up;
        cope.end = rigth_down;
        copy_and_paste_area(img, cope.start, cope.end, cope.dest_start);
    }

    if(flags.collage){
        col.start = left_up;
        col.end = rigth_down;
        col.n = n;
        col.m = m;
        collageArea(img, col.start, col.end, col.n, col.m);
    }
    writeBMPFile(output_name, img);



    return 0;

}
```

work_with_bmp.c

```
#include "work_with_bmp.h"
BMPFile* openBMPFile(char* filename) {
    FILE *filepath = fopen(filename, "rb");

    if (!filepath) {
        wprintf(L"Ошибка чтения файла \'%s\'\n", filename);
        exit(0);
    }

    BMPFile *img = (BMPFile *) malloc(sizeof(BMPFile));
```

```
    fread(&img->bmfh, sizeof(BitmapFileHeader), 1, filepath);
    fread(&img->bmih, sizeof(BitmapInfoHeader), 1, filepath);

    fclose(filepath);
    return img;
}

void readRowsBMPFile(FILE* filepath, BMPFile* img) {
    unsigned int row_size = img->bmih.width * 3;
    unsigned char row_padding = (4 - (row_size % 4)) % 4;

    unsigned char* row = (unsigned char*) malloc(row_size + row_padding);
    fseek(filepath, img->bmfh.pixelArrOffset, SEEK_SET);

    for(int y = img->bmih.height - 1; y > -1; y--){
        fread(row, row_size + row_padding, 1, filepath);
        for(int x = 0; x < img->bmih.width; x++){
            img->data[y][x].blue = row[x*3];
            img->data[y][x].green = row[x*3 + 1];
            img->data[y][x].red = row[x*3 + 2];
        }
    }
    free(row);
}

BMPFile* readBMPFile(char* filename) {
    BMPFile *img = openBMPFile(filename);

    /*if(img->bmih.compression != 0 || img->bmih.colorsInColorTable != 24 ||
img->bmfh.signature != 0x4D42){
        wprintf(L"mypaint: неподдерживаемая версия BMP-файла.\nПо
команде «--help» можно получить дополнительную информацию.\n");
        exit(0);
    }*/
```

```
    FILE *filepath = fopen(filename, "rb");
    img->data = (rgb**) malloc(img->bmih.height * sizeof(rgb*));
    for (int i = 0; i < img->bmih.height; i++)
        img->data[i] = (rgb*) malloc(img->bmih.width * sizeof(rgb));
    readRowsBMPFile(filepath, img);
    fclose(filepath);
    return img;
}

void writeBMPFile(char* filename, BMPFile* img) {
    FILE* filepath = fopen(filename, "wb");

    if (!filepath){
        wprintf(L"Ошибка чтения файла \'%s\'\n", filename);
        exit(0);
    }

    fwrite(&img->bmfh, sizeof(BitmapFileHeader), 1, filepath);
    fwrite(&img->bmih, sizeof(BitmapInfoHeader), 1, filepath);

    unsigned int row_size = img->bmih.width * 3;
    unsigned char row_padding = (4 - (row_size % 4)) % 4;

    unsigned char* row = (unsigned char*) malloc(row_size + row_padding);
    fseek(filepath, img->bmfh.pixelArrOffset, SEEK_SET);
    for (int y = img->bmih.height - 1; y > -1; y--){
        for(int x = 0; x < img->bmih.width; x++){
            row[x*3] = img->data[y][x].blue;
            row[x*3 + 1] = img->data[y][x].green;
            row[x*3 + 2] = img->data[y][x].red;
        }
        fwrite(row, row_size + row_padding, 1, filepath);
    }
```

```
    free(row);
}
void freeBMPFile(BMPFile* img) {
    if(img){
        if (img->data){
            for (int y = 0; y < img->bmih.height; y++)
                free(img->data[y]);
            free(img->data);
        }
        free(img);
    }
}

int is_bmp(char* arg){
    int length = strlen(arg);
    if (length < 4) return 0;
    char* extension = arg + length - 4;
    return strcmp(extension, ".bmp") == 0;
}
```

work_with_bmp.h

```
#include "structs.h"
BMPFile* openBMPFile(char* filename);
void readRowsBMPFile(FILE* filepath, BMPFile* img);
BMPFile* readBMPFile(char* filename);
void writeBMPFile(char* filename, BMPFile* img);
void freeBMPFile(BMPFile* img);
int is_bmp(char* arg);
```

structs.h

```
#ifndef STRUCTS_H
#define STRUCTS_H
```

```
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <locale.h>
#include <wchar.h>
#include <string.h>
#include <math.h>
#include <stdbool.h>
#include <ctype.h>
#include <regex.h>

#define PI 3.14159265

#pragma pack(push, 1)

typedef struct BitmapFileHeader{
    unsigned short signature;
    unsigned int filesize;
    unsigned short reserved1;
    unsigned short reserved2;
    unsigned int pixelArrOffset;
}BitmapFileHeader;

typedef struct BitmapInfoHeader{
    unsigned int headerSize;
    int width;
    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 RGB {
    unsigned char red;
    unsigned char green;
    unsigned char blue;
}rgb;

typedef struct BMPFile {
    BitmapFileHeader bmfh;
    BitmapInfoHeader bmih;
    rgb** data;
}BMPFile;

typedef struct coordinates{
    int x, y;
}coords;

typedef struct rect{
    coords start;
    coords end;
    unsigned int thickness;
    rgb color;
    int filled;
    rgb fill_color;
}rectangle;

typedef struct hex{
    //coords start, sec, thrd, fourth, fifth, end;
    unsigned int thickness;
    rgb color;
    int filled;
```

```
    rgb fill_color;
    coords centre;
    unsigned int radius;
}hexagon;


#pragma pack(pop)
#endif
```

functions.c

```
#include "functions.h"
//1 func
void draw_pixel(BMPFile *img, int x, int y, rgb color) {
    if (!(x < 0 || y < 0 || x >= img->bmih.width || y >= img->bmih.height)) {
        img->data[y][x] = color;
    }
}

void fill_circle(BMPFile *img, int x0, int y0, int r, rgb color) {
    int x = 0;
    int y = r;
    int delta = 3 - 2 * y;
    while (y >= x) {
        draw_pixel(img, x0 + x, y0 + y, color);
        draw_pixel(img, x0 + x, y0 - y, color);
        draw_pixel(img, x0 - x, y0 + y, color);
        draw_pixel(img, x0 - x, y0 - y, color);
        draw_pixel(img, x0 + y, y0 + x, color);
        draw_pixel(img, x0 + y, y0 - x, color);
        draw_pixel(img, x0 - y, y0 + x, color);
        draw_pixel(img, x0 - y, y0 - x, color);
        delta += delta < 0 ? 4 * x + 6 : 4 * (x - y--) + 10;
        ++x;
    }
```

```
            for (int y = -r; y <= r; y++) {
                if ((y0+y)<0 || (y0+y)>=img->bmih.height){
                    continue;
                }
                for (int x = -r; x <= r; x++) {
                    if (((x0+x)>=0) && ((x0+x)<img->bmih.width)&& (x * x + y * y <= r
* r)) {
                        draw_pixel(img, x0 + x, y0 + y, color);
                    }
                }
            }
        }

        void drawingLine(coords start, coords end, rgb color, unsigned int thickness,
BMPFile* img) {
            int x1 = start.x;
            int y1 = start.y;
            int x2 = end.x;
            int y2 = end.y;
            int dx = abs(x2 - x1);
            int dy = abs(y2 - y1);
            int sx = x1 < x2 ? 1 : -1;
            int sy = y1 < y2 ? 1 : -1;
            int err = dx - dy;
            int e2;
            int x = x1;
            int y = y1;
            while (x != x2 || y != y2) {
                draw_pixel(img,x,y,color);
                //if( (x < img->bmih.width && y < img->bmih.height && x >= 0 && y >=
0) || (x + (thickness / 2) < img->bmih.width && y + (thickness / 2) < img->bmih.height
&& x + (thickness / 2) >= 0 && y + (thickness / 2)>= 0) || (x - (thickness / 2) < img-
>bmih.width && y - (thickness / 2) < img->bmih.height && x - (thickness / 2) >= 0
&& y - (thickness / 2)>= 0)){
```

```
            if (thickness % 2 == 0) {
                fill_circle(img, x, y, thickness / 2, color);
            } else if (thickness == 1) {
                fill_circle(img, x, y, 0, color);
            } else {
                fill_circle(img, x, y, (thickness + 1) / 2, color);
            }
        //}
        e2 = 2 * err;
        if (e2 > -dy) {
            err -= dy;
            x += sx;
        }
        if (e2 < dx) {
            err += dx;
            y += sy;
        }
    }
}

    void fill_zone(int i, int j, rgb color, rgb fill_color, BMPFile* img) {//img->data массив пикселей
        if (!check_coords(i, j, img->bmih) || checkColor(img->data[i][j], color) || checkColor(img->data[i][j], fill_color)) { return; }
        img->data[i][j] = fill_color;
        fill_zone(i + 1, j, color, fill_color, img);
        fill_zone(i - 1, j, color, fill_color, img);
        fill_zone(i, j + 1, color, fill_color, img);
        fill_zone(i, j - 1, color, fill_color, img);
    }
    int isInsideArea(coords polygonPoints[], int nAngle, coords point) {
        int i, j, c = 0;
        for (i = nAngle / 3, j = nAngle - 1; i < nAngle; j = i++) {
            if (((polygonPoints[i].y > point.y) != (polygonPoints[j].y > point.y)) &&
```

```
            (point.x - 1 < (polygonPoints[j].x - polygonPoints[i].x) * (point.y - polygonPoints[i].y ) / (polygonPoints[j].y - polygonPoints[i].y) + polygonPoints[i].x))
                c = !c;
        }
        return c;
    }
    int isInsideArea1(coords polygonPoints[], int nAngle, coords point){
        int i, j, c = 0;
        for (i = 0, j = nAngle - 1; i < nAngle / 2; j = i++) {
            if (((polygonPoints[i].y > point.y) != (polygonPoints[j].y > point.y)) &&
                (point.x < (polygonPoints[j].x - polygonPoints[i].x) * (point.y - polygonPoints[i].y ) / (polygonPoints[j].y - polygonPoints[i].y) + polygonPoints[i].x ))
                c = !c;
        }
        return c;
    }
    bool checkColor(rgb a1, rgb a2){
        if(a1.blue == a2.blue && a1.green == a2.green && a1.red == a2.red){
            return true;
        } else{
            return false;
        }
    }

    bool check_coords(int y, int x, BitmapInfoHeader data){
        return !(y < 0 || y >= data.height || x < 0 || x >= data.width);
    }

    void swap(int* a, int* b)
    {
        int temp = *a;
        *a = *b;
        *b = temp;
    }
```

```
void drawingRectangle(rectangle data, BMPFile* img) {
    coords left_top = data.start;
    coords right_top = {data.start.x, data.end.y};
    coords left_bottom = {data.end.x, data.start.y};
    coords right_bottom = data.end;

    if (data.start.x > data.end.x) {
        swap(&data.start.x, &data.end.x);
    }
    if (data.start.y > data.end.y) {
        swap(&data.start.y, &data.end.y);
    }

    if (data.filled == 1) {
        for (int y = data.start.y; y < data.end.y + 1; y++) {
            for (int x = data.start.x; x < data.end.x + 1; x++) {
                draw_pixel(img, x, y, data.fill_color);
            }
        }
    }


    drawingLine(left_top, right_top, data.color, data.thickness, img);
    drawingLine(left_bottom, left_top, data.color, data.thickness, img);
    drawingLine(right_top, right_bottom, data.color, data.thickness, img);
    drawingLine(right_bottom, left_bottom, data.color, data.thickness, img);


}


void drawingHexagon(hexagon data, BMPFile* img) {
```

```
        coords l1 = {data.centre.x - (data.radius / 2), data.centre.y + data.radius *
sqrt(3.0) / 2};
        coords l2 = {data.centre.x + (data.radius / 2), data.centre.y + data.radius *
sqrt(3.0) / 2};
        coords l3 = {data.centre.x + data.radius, data.centre.y};
        coords l4 = {data.centre.x + (data.radius / 2), data.centre.y - data.radius *
sqrt(3.0) / 2};
        coords l5 = {data.centre.x - (data.radius / 2), data.centre.y - data.radius *
sqrt(3.0) / 2};
        coords l6 = {data.centre.x - data.radius, data.centre.y};
        coords polygon[6] = {l1, l2, l3, l4, l5, l6};
        if (data.filled == 1) {
           // Fill the first part of the hexagon
           for (int y = l4.y; y < l1.y; y++) {
              for (int x = l6.x; x < l3.x; x++) {
                 coords point = {x, y};
                 if (isInsideArea(polygon, 6, point)) {
                    draw_pixel(img, x, y, data.fill_color);
                 }
              }
           }
           // Fill the other part of the hexagon
           for (int y = l4.y; y < l1.y; y++) {
              for (int x = l6.x; x < l3.x; x++) {
                 coords point = {x, y};
                 if (isInsideArea1(polygon, 6, point)) {
                    draw_pixel(img, x, y, data.fill_color);
                 }
              }
           }
        }
        drawingLine(l1, l2, data.color, data.thickness, img);
        drawingLine(l2, l3, data.color, data.thickness, img);
        drawingLine(l3, l4, data.color, data.thickness, img);
```

```
            drawingLine(l4, l5, data.color, data.thickness, img);
            drawingLine(l5, l6, data.color, data.thickness, img);
            drawingLine(l6, l1, data.color, data.thickness, img);
        }
        void copy_and_paste_area(BMPFile* img, coords start, coords end, coords dest_start) {
            if (start.x > end.x) {
                swap(&start.x, &end.x);
            }
            if (start.y > end.y) {
                swap(&start.y, &end.y);
            }
            if(start.x < 0) start.x = 0;
            if(start.y < 0) start.y = 0;
            if(start.x >= img->bmih.width) start.x = img->bmih.width - 1;
            if(start.y >= img->bmih.height) start.y = img->bmih.height - 1;

            if(end.x < 0) end.x = 0;
            if(end.y < 0) end.y = 0;
            if(end.x >= img->bmih.width) end.x = img->bmih.width - 1;
            if(end.y >= img->bmih.height) end.y = img->bmih.height - 1;

            if(dest_start.x < 0) dest_start.x = 0;
            if(dest_start.y < 0) dest_start.y = 0;
            if(dest_start.x >= img->bmih.width) dest_start.x = img->bmih.width - 1;
            if(dest_start.y >= img->bmih.height) dest_start.y = img->bmih.height - 1;

            int width = end.x - start.x + 1;
            int height = end.y - start.y + 1;

            //printf("%d.%d - %d.%d\n%d.%d", start.x, start.y, end.x, end.y, dest_start.x, dest_start.y);
            // Создаем временный буфер для хранения скопированной области
            rgb** temp_buffer = (rgb**)malloc(sizeof(rgb*) * height);
```

```
        if (!temp_buffer) {
            printf("Memory allocation failed.\n");
            return;
        }
        for (int i = 0; i < height; i++) {
            temp_buffer[i] = (rgb*)malloc(sizeof(rgb) * width);
            if (!temp_buffer[i]) {
                printf("Memory allocation failed.\n");
                // Освобождаем ранее выделенную память
                for (int j = 0; j < i; j++) {
                    free(temp_buffer[j]);
                }
                free(temp_buffer);
                return;
            }
        }

        // Копируем область изображения во временный буфер
        for (int y = start.y; y < end.y; y++) {
            for (int x = start.x; x < end.x; x++) {
                if( y < img->bmih.height &&  x < img->bmih.width)   {
                    temp_buffer[y - start.y][x - start.x] = img->data[y][x];
                }
            }

        }

        // Вставляем скопированную область в указанное место в изображении
        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                if(dest_start.y + y < img->bmih.height && dest_start.x + x < img-
>bmih.width)   {
                    img->data[dest_start.y + y][dest_start.x + x] = temp_buffer[y][x];
                }
```

```
        }
    }

    // Освобождаем временный буфер
    for (int i = 0; i < height; i++) {
        free(temp_buffer[i]);
    }
    free(temp_buffer);
}

void collageArea(BMPFile* img, coords start, coords end, int n, int m) {
    int razn_y = end.y - start.y;
    int razn_x = end.x - start.x;

    if (razn_y <= 0 || razn_x <= 0) {
        printf("Invalid coordinates for the area.\n");
        return;
    }

    rgb** temp_buffer = (rgb**)malloc(sizeof(rgb*) * razn_y);
    if (!temp_buffer) {
        printf("Memory allocation failed.\n");
        return;
    }
    for (int i = 0; i < razn_y; i++) {
        temp_buffer[i] = (rgb*)malloc(sizeof(rgb) * razn_x);
        if (!temp_buffer[i]) {
            printf("Memory allocation failed.\n");
            for (int j = 0; j < i; j++) {
                free(temp_buffer[j]);
            }
            free(temp_buffer);
            return;
        }
```

```
        }

        // Копирование выбранной области в временный буфер
        for (int y = 0; y < razn_y; y++) {
            for (int x = 0; x < razn_x; x++) {
                if (start.y + y < img->bmih.height && start.x + x < img->bmih.width) {
                    temp_buffer[y][x] = img->data[start.y + y][start.x + x];
                }
            }
        }

        int countN = 0;
        int countM = 0;
        // Повторение узора по всей области изображения
        for (int y = 0; y < img->bmih.height; y++) {
            for (int x = 0; x < img->bmih.width; x++) {
                img->data[y][x] = temp_buffer[y % razn_y][x % razn_x];
            }
        }

        printf("%d--%d", countN, countM);
        // Освобождение временного буфера
        for (int i = 0; i < razn_y; i++) {
            free(temp_buffer[i]);
        }
        free(temp_buffer);
    }
```

functions.h

```
#include "structs.h"
void draw_pixel(BMPFile *img, int x, int y, rgb color);
void fill_circle(BMPFile *img, int x0, int y0, int r, rgb color);
```

```
void drawingLine(coords start, coords end, rgb color, unsigned int thickness, BMPFile* img);
bool checkColor(rgb a1, rgb a2);
bool check_coords(int y, int x, BitmapInfoHeader data);
void fill_zone(int i, int j, rgb color, rgb fill_color, BMPFile* img);
int isInsideArea(coords polygonPoints[], int nAngle, coords point);
int isInsideArea1(coords polygonPoints[], int nAngle, coords point);
void fill_rectangle(BMPFile *img, int x0, int y0, rgb color);
void drawingRectangle(rectangle data, BMPFile* img);
void drawingHexagon(hexagon data, BMPFile* img);
void copy_and_paste_area(BMPFile* img, coords start, coords end, coords past_in_area);
void collageArea(BMPFile* img, coords start, coords end, int n, int m);
```

Makefile

```
FLAGS = -std=c99

all: cw

cw: main.o functions.o work_with_bmp.o checkers.o
	gcc -o cw $^ -lm

main.o: main.c structs.h functions.h work_with_bmp.h checkers.h
	gcc -c $<

functions.o: functions.c structs.h functions.h
	gcc -c $<

work_with_bmp.o: work_with_bmp.c structs.h work_with_bmp.h
	gcc -c $<

checkers.o: checkers.c structs.h checkers.h
	gcc -c $<
```

```
clean:
	rm -f *.o cw
```