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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка bmp изображений

Студентка гр. 2381 _____ Слабнова Д.А.

Преподаватель _____ Тиняков С.А.

Санкт-Петербург

2023

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студентка Слабнова Д.А.

Группа 2381

Тема работы: обработка bmp изображений

Исходные данные:
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:

- Рисование правильного шестиугольника. Шестиугольник определяется:
- либо координатами левого верхнего и правого нижнего угла квадрата, в который он вписан, либо координатами его центра и радиусом в который он списан, толщиной линий, цветом линий. Шестиугольник может быть залит или нет. Цветом которым залит шестиугольник, если пользователем выбран залитый.
- Копирование заданной области. Функционал определяется: координатами

левого верхнего угла области-источника, координатами правого нижнего угла области-источника, координатами левого верхнего угла области-назначения.

- Заменяет все пиксели одного заданного цвета на другой цвет. Функционал определяется: цвет, который требуется заменить, цвет на который требуется заменить.
- Сделать рамку в виде узора. Рамка определяется: узором должно быть несколько на выбор. Красивый узор можно получить, используя фракталы, цветом, шириной.

Содержание пояснительной записки:
«Содержание», «Введение», «Заключение», «Список использованных источников»

Предполагаемый объем пояснительной записки:
Не менее 40 страниц.

Дата выдачи задания: 19.03.2023

Дата сдачи реферата: 17.05.2023

Дата защиты реферата: 19.05.2023

Студентка _____ Слабнова Д.А.

Преподаватель _____ Тиняков С.А.

АННОТАЦИЯ

В курсовой работе требуется реализовать программу по обработке изображений формата bmp. Программа должна быть способной рисовать шестиугольник на заданном bmp файле, копировать участок одного bmp файла на другой, поменять все пиксели определённого цвета на новый цвет и рисовать рамку для заданного bmp файла. Программа должна иметь CLI (command line interface). Для написания кода используются стандартные библиотеки языка Си, а также библиотека getopt.h.

SUMMARY

In the course work, it is required to implement a program for processing images of the bmp format. The program should be able to draw a hexagon on a given bmp file, copy a section of one bmp file to another, change all pixels of a certain color to a new color, and draw a frame for a given bmp file. The program must have CLI (command line interface). To write the code, standard C libraries are used, as well as the getopt.h library.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы – написать программу на языке Си для обработки bmp изображений. Также полученная программа должна иметь консольный интерфейс.

1. ЧТЕНИЕ И ЗАПИСЬ BMP ФАЙЛА.

1.1. структура bmpheader, bmpinfheader и bmp

BMP файл состоит из трёх частей: 1)header(базовая информация о файле) – первые 14 бит файла, содержат сигнатуру (чтобы файл можно было распознать как bmp), размер всего файла, два поля reserved, значение которых всегда 0 и офсет – начальный адрес массива самих пикселей изображения. 2)DIB header(более подробная информация о файле) – может быть 40, 108, 124 бита, в зависимости от версии файла. Для файлов с третей по пятую версию dib header содержит информацию о размере dib header, ширине и высоте изображения, количеству бит в цвете, информацию о методе сжатия, количество цветов и важных цветов (зачастую эти параметры игнорируют и их значение равно нулю), а также количество пикселей на метр по y и по x. Для работы с bmp были реализованы структуры HEADERFILE и HEADERINFO – которые хранят информацию из header и DIB header соответственно. Структура BMP содержит поля head и inf – header и DIB header соответствующего bmp файла. Поле version хранит номер версии. Поле padding_bytes хранит количество байт – нулей, которые нужно дописать в конец ряда (длина ряда пикселей в байтах должна делиться на 4, в случае, когда это не так, в конец дописываются лишние нули). Поле arr – хранит указатель на двумерный массив пикселей(пиксель – структура RGB, имеющая поля r, g, b для соответствующих компонент).

1.2. Функции get_img(), put_img(), free_bmp()

Функция get_img получает на вход путь к файлу из текущей директории, в которой запущена программа, и возвращает структуру BMP, хранящую информацию о выбранном файле(поле inf и head соответствуют header и DIB header файла). Функция создаёт двумерный динамический массив для хранения пикселей и присваивает полю arr возвращаемой структуры указатель на данный массив. Функция вычисляет значение padding_bytes, а также присваивает значение полю version на основе информации о размере DIB header.

Функция put_img принимает на вход имя файла для записи и структуру BMP с информацией для записи. Функция записывает информацию из полей head и inf в файл (с помощью pragma pack(push, 1), структуры «упакованы» так, что байты информации идут друг за другом, как и требуется в файле). Затем построчно функция записывает элементы arr в файл и добавляет padding_bytes байт в конец строки.

Функция free_BMP освобождает память, занятую массивом пикселей.

2. ОСНОВНЫЕ ФУНКЦИИ

2.1. Изменение цвета. Change_color().

Вспомогательная функция change_clr принимает указатель на RGB и RGB, и меняет значение цвета по указателю на второй аргумент. Функция cmp_rgb сравнивает RGB, и возвращает ненулевое значение, если все их поля совпадают.

Функция change_color принимает на вход имя файла, и RGB на замену и RGB, на который надо заменить. Функция проходит по всем пикселям картинки, проверяя их на равенство заменяемому цвету, и если они равны заменяемому, меняет их цвет.

2.2. Копирование и вставка. Copy_paste().

Функция получает на вход имя файла для копирования, файла для вставки, координаты правого нижнего и левого верхнего углов копируемого фрагмента и левого верхнего угла области назначения. Функция, проходя с помощью циклов for по выделенному фрагменту, меняет цвет соответствующего пикселя на изображении для вставки. Таким образом, на изображении для вставки «рисуется» выделенный фрагмент.

2.3Шестиугольник. draw_hexagon().

Вспомогательная функция lin_func, принимает на вход коэффициенты линейного уравнения y=b*x+c и значение x и возвращает значение y. Функция draw_line использует функцию lin_func для рисования непрерывных прямых линий (у любого пикселя линий один из 8 его соседей тоже принадлежит линии), принимая на вход координаты начала и конца линии, её цвета и BMP структуры, на которой требуется нарисовать линию. Функция draw_thick_line, с помощью функции draw_line может рисовать толстые лини и , в зависимости от переданного флага, может «утолщать» линию вверх или вниз, направо или налево (принимает на вход BMP, цвет линии, координаты начала и конца, толщину, направление утолщения).

Функция draw_hexagon в зависимости от переданных ей аргументов, вычисляет координаты вершин правильного шестиугольника, затем рисует с помощью draw_thick_line грани фигуры, и если переданный ей флаг fill_flag ненулевой, закрашивает его, с помощью цикла for проходя по всем пикселям области между краями шестиугольника, и перекрашивая их.

2.4Рамка. draw_..._frame().

Было реализовано три вида рамок.

Кривая Коха – фрактальная кривая, описанная в 1904 году шведским математиком Хельге фон Кохом. Процесс её построения выглядит следующим образом: берём единичный отрезок, разделяем на три равные части и заменяем средний интервал равносторонним треугольником без этого сегмента. В результате образуется ломаная, состоящая из четырёх звеньев длины 1/3. На следующем шаге повторяем операцию для каждого из четырёх получившихся звеньев и т. д… Предельная кривая и есть кривая Коха. Функция draw_Koch_snowflake принимает на вход координаты начала и конца отрезка, на котором требуется нарисовать кривую, цвет, порядок и BMP, на котором требуется рисовать. Функция рассчитывает координаты звеньев линии и вызывает саму себя от полученных фрагментов с меньшим на единицу порядком. При порядке равном 0 рисует прямую с помощью draw_line. Функция draw_Koch_snowflake получает на вход имя файла, толщину рамки, цвет линий, флаг, залить рамку или нет, и цвет заливки. Функция рисует на изображении кривые вдоль границ рамки, и если флаг заливки ненулевой, заливает её.

Кривая Минковского - геометрический фрактал, предложенный Минковским - немецким математиком. Процесс построения см. рис. 2.3.

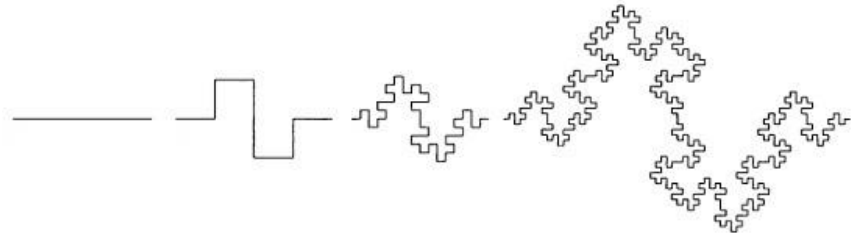


Рис. 2.3.

Функция draw_Minkowski_sausage принимает на вход BMP, координаты начала и конца отрезка, цвет линий и порядок. Функция вычесляет координаты звеньев и вызывает саму себя от отрезков полученной ломанной с порядком, меньшим на единицу. Если порядок равен нулю, функция рисует прямую линию. Функция draw_Minkowski_frame получает на вход имя файла, толщину рамки, цвет линий, флаг, залить рамку или нет, и цвет заливки. Функция рисует на изображении кривые вдоль границ рамки, и если флаг заливки ненулевой, заливает её.

Функция draw_Simple_frame получает на вход имя файла, цвет и толщину линий и, с помощью draw_thick_line рисует простую рамку в виде цветных полос по краям изображения.

2.5. Вспомогательные функции.

Чтобы проверить, можно ли использовать переданные координаты в функцию (не выходят ли координаты за пределы массива и т.п.), каждая функция использует вспомогательные функции, названия которых начинаются с check_.... Функции b() и c() принимают на вход координаты двух точек и возвращают коэффициенты b и c линейного уравнения y=b*x+c для прямой, проходящей через данные точки.

Также был реализован алгоритм заливки fill_v2 (применяется для заливки рамки), который закрашивает область ограниченную краями рисунка и непрерывной линией одного цвета. Алгоритм: на вход подаётся проверяемый пиксель, если его нужно закрасить (он не цвета границы и он не вне изображения), закрашиваем и вызываем алгоритм от четырёх его соседей (слева, сверху, справа, снизу). Чтобы избежать stackoverflow для хранения пикселей на проверку была реализована структура – очередь (односвязный список) и соответствующие функции вставки в конец и удаления начала. До тех пор пока очередь не пуста, проделываем описанные выше действия над головой очереди, «соседей» на проверку вставляем в конец очереди.

3.ПРОЧИЕ ФУНКЦИИ

3.1 main.c и вспомогательные функции.

Main.c может принимать на вход аргументы, но при их отсутствии выводит справку к программе. С помощью библиотеки getopt.h main обрабатывает полученные на вход аргументы. Функция printhelp выводит справку. Функция printinf получает на вход имя файла и выводит информацию о нём. Функция check_cord проверяет, находится ли левая верхняя координата левее и выше правой нижней координаты. Функция not_num_check принимает на вход строку и возвращает ненулевое значение, если она содержит не натуральное число.

ЗАКЛЮЧЕНИЕ

Была написана программа на языке Си, обрабатывающая bmp изображения и использующая CLI – command line interface.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

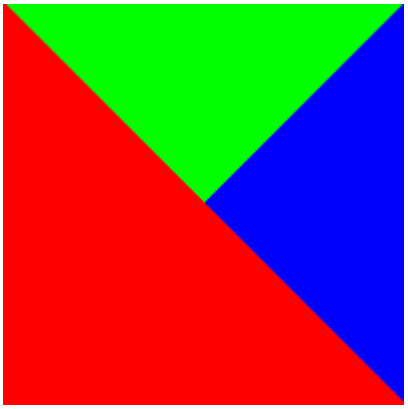
1. Брайан Керниган, Денис Ритчи «язык программирования Си». 2001.
2. Сайт Purdue university // https://engineering.purdue.edu/ece264/17au/hw/HW15
3. Документация майкрософт / / https://learn.microsoft.com/en-us/windows/win32/api/wingdi/ns-wingdi-bitmap

ПРИЛОЖЕНИЕ А

ПРИМЕР РАБОТЫ ПРОГРАММЫ

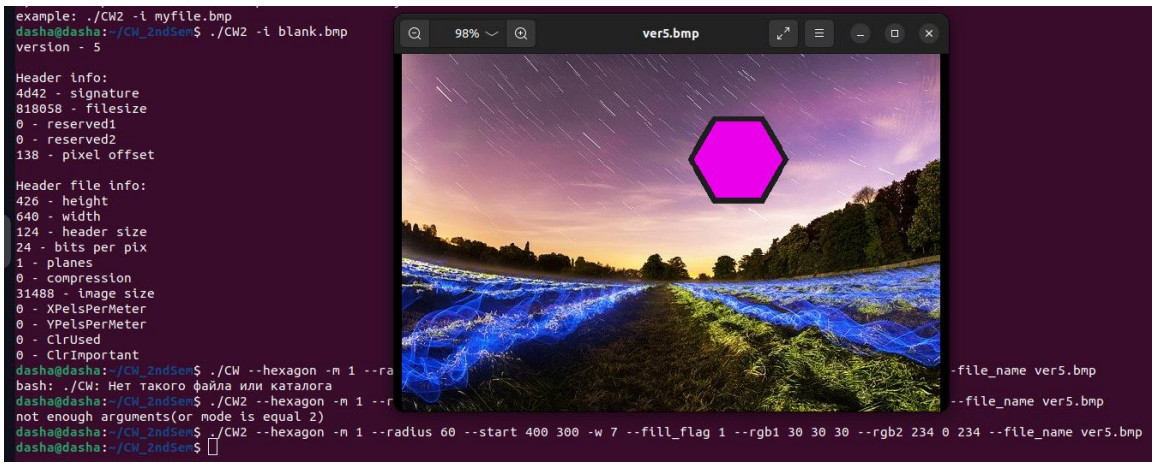
Blank.bmp до изменений

Ver3.bmp до изменений

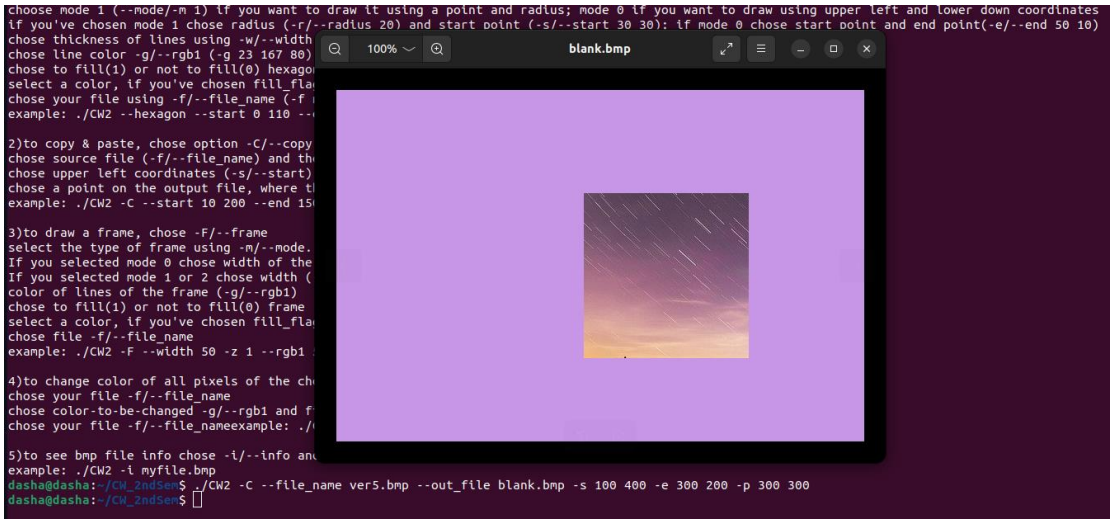


Ver5.bmp до изменений

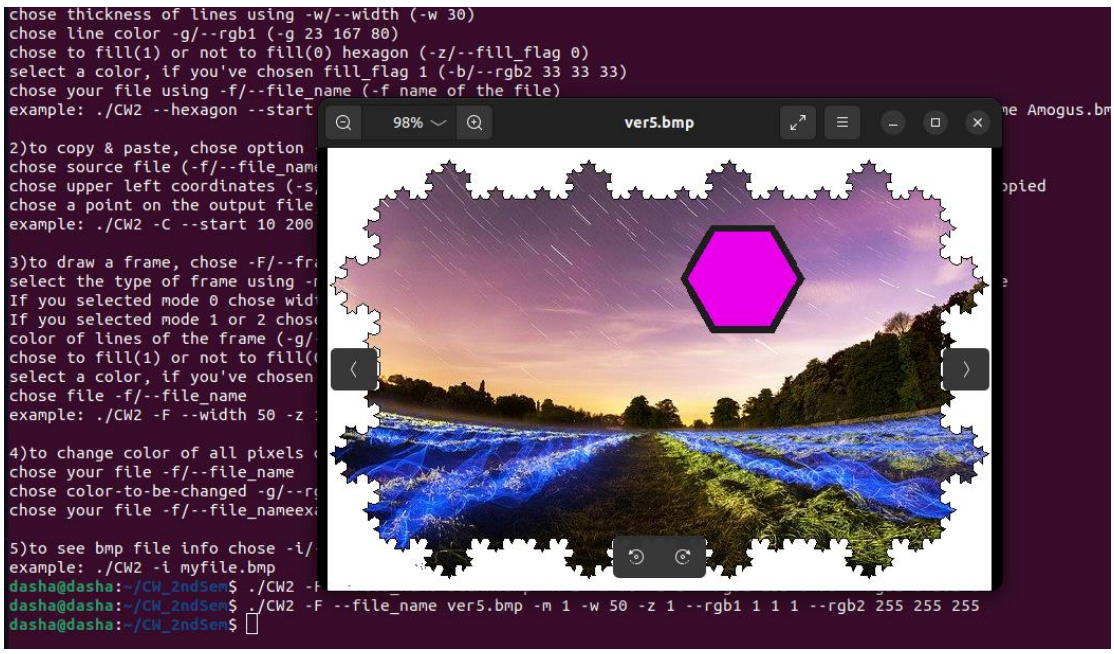
Пример рисования шестиугольника



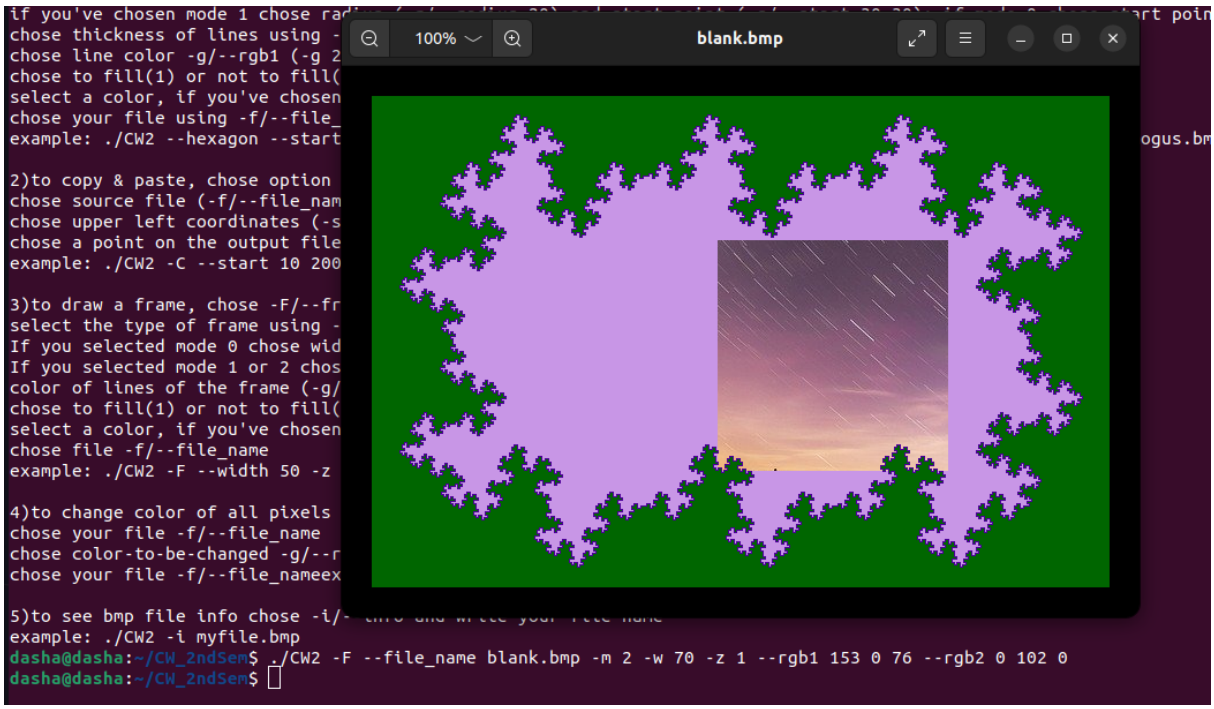
Пример копирования



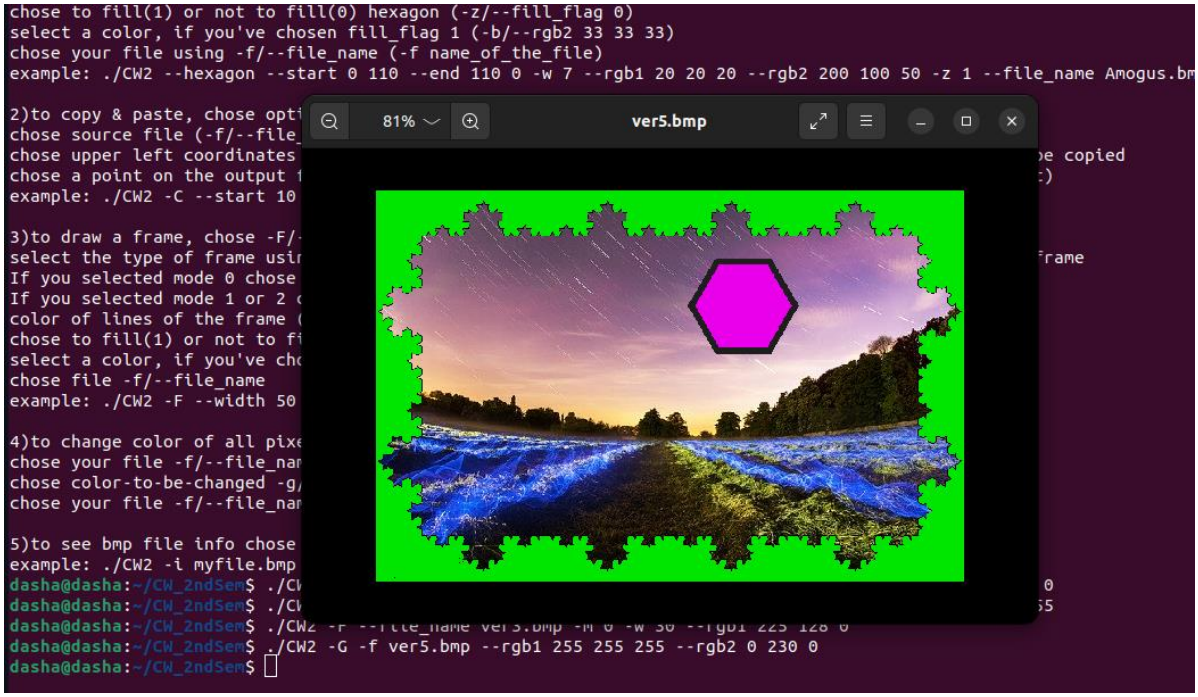
Пример рисования рамки



Пример рисования рамки 2



Пример смены цвета



Пример вывода справки

```
dasha@dasha:~/CW_2ndSem$ ls
blank.bmp  cmake-build-debug  CMakeLists.txt  CW2  CWlib.h  frame.c  func.c  func.h  main.c  Makefile  read_write.c  read_write.h  samples  ver3.bmp  ver5.bmp
dasha@dasha:~/CW_2ndSem$ ./CW2
You have 5 options:
1)draw Hexagon, 2)copy a part of an image and paste it on another image 3)draw a frame 4)change chosen color on other color 5)show bmp file info

1)to draw hexagon, choose option -H/--hexagon
choose mode 1 (--mode/-m 1) if you want to draw it using a point and radius; mode 0 if you want to draw using upper left and lower down coordinates
if you've chosen mode 1 chose radius (-r/--radius 20) and start point (-s/--start 30 30); if mode 0 chose start point and end point(-e/--end 50 10)
chose thickness of lines using -w/--width (-w 30)
chose line color -g/--rgb1 (-g 23 167 80)
chose to fill(1) or not to fill(0) hexagon (-z/--fill_flag 0)
select a color, if you've chosen fill_flag 1 (-b/--rgb2 33 33 33)
chose your file using -f/--file_name (-f name_of_the_file)
example: ./CW2 --hexagon --start 0 110 --end 110 0 -w 7 --rgb1 20 20 20 --rgb2 200 100 50 -z 1 --file_name Amogus.bmp

2)to copy & paste, chose option -C/--copy
chose source file (-f/--file_name) and the file, where you going to paste to, (-o/--out_file)
chose upper left coordinates (-s/--start) and lower right (-e/--end) of the part of source image to be copied
chose a point on the output file, where the upper left corner of the copied piece will be (-p/--point)
example: ./CW2 -C --start 10 200 --end 150 40 -p 30 230 --file_name file1.bmp --out_file file2.bmp

3)to draw a frame, chose -F/--frame
select the type of frame using -m/--mode. 0 - simple frame, 1 - Koch line frame, 2 - Minkowski line frame
If you selected mode 0 chose width of the frame (-w/--width) and color (-g/--rgb1) only.
If you selected mode 1 or 2 chose width (-w/--width) of the frame,
color of lines of the frame (-g/--rgb1)
chose to fill(1) or not to fill(0) frame
select a color, if you've chosen fill_flag 1 (-b/--rgb2)
chose file -f/--file_name
example: ./CW2 -F --width 50 -z 1 --rgb1 55 55 55 --rgb2 1 1 1 -f file.bmp

4)to change color of all pixels of the chosen color to the other color, choose option -G/--changeclr
chose your file -f/--file_name
chose color-to-be-changed -g/--rgb1 and finish color -b/--rgb2
chose your file -f/--file_nameexample: ./CW2 -G --rgb1 45 45 45 --rgb2 88 88 88 -f myfile.bmp

5)to see bmp file info chose -i/--info and write your file name
example: ./CW2 -i myfile.bmp
dasha@dasha:~/CW_2ndSem$
```

Пример вывода информации о файле

```
dasha@dasha:~/CW_2ndSem$ ./CW2 -i blank.bmp
version - 5

Header info:
4d42 - signature
818058 - filesize
0 - reserved1
0 - reserved2
138 - pixel offset

Header file info:
426 - height
640 - width
124 - header size
24 - bits per pix
1 - planes
0 - compression
31488 - image size
0 - XPelsPerMeter
0 - YPelsPerMeter
0 - ClrUsed
0 - ClrImportant
dasha@dasha:~/CW_2ndSem$
```

Пример обработки ошибок

```
dasha@dasha:~/CW_2ndSem$ ./CW2 --jinx
unknown option found.
dasha@dasha:~/CW_2ndSem$ ./CW2 --rgb 45 7
unknown option found.
dasha@dasha:~/CW_2ndSem$ ./CW2 --rgb1 45 7
not enough arguments
dasha@dasha:~/CW_2ndSem$ ./CW2 --rgb1 45 300 23
Impossible color! r, g, b should be less than 256
dasha@dasha:~/CW_2ndSem$ ./CW2 -H -f blank.bmp --rgb1 240 130 20 -m 0 -z 0 -w 20 --start 1000 1000 --end 300 0
wrong order of coordinates
dasha@dasha:~/CW_2ndSem$ ./CW2 -H -f blank.bmp --rgb1 240 130 20 -m 0 -z 0 -w 20 --start 0 300 --end 1000 0
impossible coordinates
```

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

```
#include "CWlib.h"
#include "read_write.h"
#include "func.h"
#include "getopt.h"


int main(int argc, char *argv[]){
    RGB clr1, clr2;
    int r, g, b;
    int clr1_is_taken_flag = 0; int clr2_is_taken_flag = 0;
    int width = -1; int radius = -1;
    int data[4];
    int xp = -1; int yp = -1;
    int data_len = 0;
    int flag_opt_choosen = 0;
    char option = 'h';
    char* file_name = NULL; char* output_file = NULL;
    int mode = -1; int fill_flag = -1;
    int some_index;

    opterr = 0;
    char* optstring = "hHCGFs:e:p:r:g:w:b:f:o:m:z:i:";
    struct option longg[] = {
            {"help",      no_argument, NULL, 'h'},
            {"hexagon",   no_argument, NULL, 'H'},
            {"copy",      no_argument, NULL, 'C'},
            {"changeclr", no_argument, NULL, 'G'},
            {"frame", no_argument, NULL, 'F'},
            {"start", required_argument, NULL, 's'},
            {"end", required_argument, NULL, 'e'},
            {"point", required_argument, NULL, 'p'},
            {"radius", required_argument, NULL, 'r'},
            {"rgb1", required_argument, NULL, 'g'},
            {"rgb2", required_argument, NULL, 'b'},
            {"width", required_argument, NULL, 'w'},
            {"file_name", required_argument, NULL, 'f'},
            {"out_file", required_argument, NULL, 'o'},
            {"mode", required_argument, NULL, 'm'},
            {"fill_flag", required_argument, NULL, 'z'},
            {"info", required_argument, NULL, 'i'},
            {0, 0, 0, 0}
    };
    int opt;
    while((opt = getopt_long(argc, argv, optstring, longg, &some_index))
!= -1){
        switch(opt){
            case 'h':
                if(!flag_opt_choosen){
                    option = 'h';
                    flag_opt_choosen = 1;
                }else{
```

```
                    printf("you cant do two things at the same time. You
are not Caesar.\n");
                    exit(1);
                }
                break;
            case 'H':
                if(!flag_opt_choosen){
                    option = 'H';
                    flag_opt_choosen = 1;
                }else{
                    printf("you cant do two things at the same time. You
are not Caesar.\n");
                    exit(1);
                }
                break;
            case 'C':
                if(!flag_opt_choosen){
                    option = 'C';
                    flag_opt_choosen = 1;
                }else{
                    printf("you cant do two things at the same time. You
are not Caesar.\n");
                    exit(1);
                }
                break;
            case 'G':
                if(!flag_opt_choosen){
                    option = 'G';
                    flag_opt_choosen = 1;
                }else{
                    printf("you cant do two things at the same time. You
are not Caesar.\n");
                    exit(1);
                }
                break;
            case 'F':
                if(!flag_opt_choosen){
                    option = 'F';
                    flag_opt_choosen = 1;
                }else{
                    printf("you cant do two things at the same time. You
are not Caesar.\n");
                    exit(1);
                }
                break;
            case 's':
                if(optind >= argc){
                    printf("not enough arguments\n");
                    exit(1);
                }
                if(not_num_check(optarg) || not_num_check(argv[optind])){
                    printf("There is an impostor! Not Natural number
found.\n");
                    exit(1);
                }else{
                    data_len += 2;
                    data[0] = atoi(optarg);
```

```
                    data[1] = atoi(argv[optind]);
                }
                break;
            case 'e':
                if(optind >= argc){
                    printf("not enough arguments\n");
                    exit(1);
                }
                if(not_num_check(optarg) || not_num_check(argv[optind])){
                    printf("There is an impostor! Not Natural number
found.\n");
                    exit(1);
                }else{
                    data_len += 2;
                    data[2] = atoi(optarg);
                    data[3] = atoi(argv[optind]);
                }
                break;
            case 'p':
                if(optind >= argc){
                    printf("not enough arguments\n");
                    exit(1);
                }
                if(not_num_check(optarg) || not_num_check(argv[optind])){
                    printf("There is an impostor! Not Natural number
found.\n");
                    exit(1);
                }else{
                    xp = atoi(optarg);
                    yp = atoi(argv[optind]);
                }
                break;
            case 'r':
                if(not_num_check(optarg)){
                    printf("There is an impostor! Not Natural number
found.\n");
                    exit(1);
                }else{
                    radius = atoi(optarg);
                }
                break;
            case 'g':
                if(optind+1 >= argc){
                    printf("not enough arguments\n");
                    exit(1);
                }
                if(clr1_is_taken_flag){
                    printf("you cant have two colors in one\n");
                    exit(1);
                }
                if(not_num_check(optarg) || not_num_check(argv[optind])
|| not_num_check(argv[optind+1])){
                    printf("There is an impostor! Not Natural number
found.\n");
                    exit(1);
                }else{
                    r = atoi(optarg);
```

```
                    g = atoi(argv[optind]);
                    b = atoi(argv[optind+1]);
                    if(r > 255 || g > 255 || b > 255){
                        printf("Impossible color! r, g, b should be less
than 256\n");
                        exit(1);
                    }
                    clr1.r = r;
                    clr1.g = g;
                    clr1.b = b;
                }
                clr1_is_taken_flag = 1;
                break;
            case 'b':
                if(optind+1 >= argc){
                    printf("not enough arguments\n");
                    exit(1);
                }
                if(clr2_is_taken_flag){
                    printf("you cant have two colors in one\n");
                    exit(1);
                }
                if(not_num_check(optarg)  ||  not_num_check(argv[optind])
|| not_num_check(argv[optind+1])){
                    printf("There  is  an  impostor!  Not  Natural  number
found.\n");
                    exit(1);
                }else{
                    r = atoi(optarg);
                    g = atoi(argv[optind]);
                    b = atoi(argv[optind+1]);
                    if(r > 255 || g > 255 || b > 255){
                        printf("Impossible color! r, g, b should be less
than 256\n");
                        exit(1);
                    }
                    clr2.r = r;
                    clr2.g = g;
                    clr2.b = b;
                }
                clr2_is_taken_flag = 1;
                break;
            case 'w':
                if(not_num_check(optarg)){
                    printf("There  is  an  impostor!  Not  Natural  number
found.\n");
                    exit(1);
                }else{
                    width = atoi(optarg);
                }
                break;
            case 'f':
                file_name = optarg;
                break;
            case 'o':
                output_file = optarg;
                break;
```

```
            case 'm':
                if(not_num_check(optarg)){
                    printf("There is an impostor! Not Natural number
found.\n");
                    exit(1);
                }else{
                    mode = atoi(optarg);
                    if(mode != 0 && mode != 1 && mode != 2){
                        printf("mode can only be 2, 1 or 0\n");
                        exit(1);
                    }
                }
                break;
            case 'z':
                if(not_num_check(optarg)){
                    printf("There is an impostor! Not Natural number
found.\n");
                    exit(1);
                }else{
                    fill_flag = atoi(optarg);
                    if(fill_flag != 0 && fill_flag != 1){
                        printf("fill flag can only be 1 or 0\n");
                        exit(1);
                    }
                }
                break;
            case 'i':
                printinf(optarg);
                return 0;
            case '?':
                printf("unknown option found.\n");
                return 0;

        }
    }
    if(data_len > 4){
        printf("why so many data?\n");
        exit(1);
    }
    switch(option){
        case 'h':
            printhelp();
            break;
        case 'H':
            if(mode == 1){
                if(radius == -1){
                    printf("not enough arguments\n");
                    exit(1);
                }
                data[2] = data[1];
                data[1] = data[0];
                data[0] = radius;
            }
            if(fill_flag == -1 || fill_flag == 0){
                clr2_is_taken_flag = 1;
                clr2.r = 0; clr2.b = 0; clr2.g = 0;
            }
```

```
            if(!file_name || data_len < 2 || mode == -1 || mode == 2 ||
width == -1 || !clr2_is_taken_flag || !clr2_is_taken_flag){
                printf("not enough arguments(or mode is equal 2)\n");
                exit(1);
            }
            if(mode == 0){
                if(!check_cord(data[0], data[1], data[2], data[3])){
                    printf("wrong order of coordinates\n");
                    exit(1);
                }
            }
            draw_hexagon(file_name, data, mode, width, clr1, fill_flag,
clr2);
            break;
        case 'C':
            if(!file_name || !output_file || data_len < 4 || xp == -1 ||
yp == -1){
                printf("not enough arguments\n");
                exit(1);
            }
            if(!check_cord(data[0], data[1], data[2], data[3])){
                printf("wrong order of coordinates\n");
                exit(1);
            }
            copy_paste(file_name, output_file, data[0], data[1], data[2],
data[3], xp, yp);
            break;
        case 'G':
            if(!file_name || !clr2_is_taken_flag || !clr1_is_taken_flag){
                printf("not enough arguments\n");
                exit(1);
            }
            change_color(file_name, clr1, clr2);
            break;
        case 'F':
            if(!file_name || !width || !clr1_is_taken_flag){
                printf("not enough arguments\n");
                exit(1);
            }
            if(mode == 0){
                draw_simple_frame(file_name, width, clr1);
            }else if(mode == 1){
                if(fill_flag == -1 || !clr2_is_taken_flag){
                    printf("not enough arguments\n");
                    exit(1);
                }
                draw_Koch_frame(file_name, width, fill_flag, clr1, clr2);
            }else{
                if(fill_flag == -1 || !clr2_is_taken_flag){
                    printf("not enough arguments\n");
                    exit(1);
                }
                draw_Minkowski_frame(file_name, width, fill_flag, clr1,
clr2);
            }
            break;
    }
```

```
    return 0;
}
```

Название файла: read_write.c

```
#include "CWlib.h"
#include "func.h"
#include <ctype.h>
#include <string.h>
void free_BMP(BMP bmp){
    for(int i = 0; i < abs(bmp.inf.Height); i++){
        free((bmp.arr)[i]);
    }
    free(bmp.arr);
}

BMP get_img(char* file_name){
    FILE* fp = fopen(file_name, "rb"); //r - read, b - binary
    if(fp == NULL){
        printf("couldn't open %s file in get_img. Does it actually
exist?\n", file_name);
        exit(1);
    }
    HEADERFILE header;
    HEADERINFO headinf;
    BMP ans;

    fread(&header, 1, sizeof(HEADERFILE), fp);
    DWORD headSize; //to find out which version

    fread(&headSize, 1, sizeof(DWORD), fp);
    headinf.Size = headSize;
    fread(&(headinf.Width), 1, headinf.Size - sizeof(DWORD), fp);

    ans.head = header;
    ans.inf = headinf;

    ans.version = 0;

    if(headSize == 40){
        ans.version = 3;
    }else if(headSize == 108){
        ans.version = 4;
    }else if(headSize == 124){
        ans.version = 5;
    }


    int height = abs(headinf.Height);
    int width = headinf.Width;

    int padding = (width*sizeof(RGB))%4; // padding to be added
    BYTE trash[4]; //put your extra padding here

    RGB** arr = malloc(sizeof(RGB*)*height);
    if(arr == NULL){
        printf("couldn't do malloc in function get_img.\n");
        exit(1);
    }
```

```
    for(int i = 0; i < height; i++){
        arr[i] = malloc(sizeof(RGB) * width);
        if(arr[i] == NULL){
            printf("couldn't do malloc in function get_img.\n");
            exit(1);
        }
        fread(arr[i], sizeof(RGB), width, fp);
        fread(trash, 1, padding, fp);
    }
    ans.arr = arr;
    fclose(fp);

    ans.padding_bytes = padding;

    return ans;
}

void put_img(char* file_name, BMP bmp) {
    FILE *fp = fopen(file_name, "wb");
    if(fp == NULL){
        printf("couldn't open %s file in put_img.\n", file_name);
        exit(1);
    }
    fwrite(&(bmp.head), 1, sizeof(HEADERFILE), fp);
    fwrite(&(bmp.inf), 1, bmp.inf.Size, fp);
    int height = abs(bmp.inf.Height);
    int width = bmp.inf.Width;
    BYTE *padding = calloc(bmp.padding_bytes, sizeof(BYTE));
    if(padding == NULL){
        printf("couldn't do calloc in function put_img.\n");
        exit(1);
    }
    for (int i = 0; i < height; i++) {
        fwrite((bmp.arr)[i], sizeof(RGB), width, fp);
        fwrite(padding, sizeof(BYTE), bmp.padding_bytes, fp);
    }
    fclose(fp);

}

int not_num_check(char* s){ //check if it's a non-zero positive integer
    if(!strlen(s)) return 0;
    if(s[0] == '0' && (strlen(s) > 1)) return 0;
    int i = 0;
    while(s[i] != '\0'){
        if(!isdigit(s[i])){
            return 1;
        }
        i++;
    }
    return 0;
};

void printhelp(){
```

```
    printf("You have 5 options:\n1)draw Hexagon, 2)copy a part of an
image and paste it on another image 3)draw a frame 4)change chosen color
on other color 5)show bmp file info\n");
    printf("\n");
    printf("1)to draw hexagon, choose option -H/--hexagon\nchoose mode 1
(--mode/-m 1) if you want to draw it using a point and radius; mode 0 if
you want to draw using upper left and lower down coordinates\n");
    printf("if you've chosen mode 1 chose radius (-r/--radius 20) and
start point (-s/--start 30 30); if mode 0 chose start point and end
point(-e/--end 50 10)\n");
    printf("chose thickness of lines using -w/--width (-w 30)\n");
    printf("chose line color -g/--rgb1 (-g 23 167 80)\n");
    printf("chose to fill(1) or not to fill(0) hexagon (-z/--fill_flag
0)\n");
    printf("select a color, if you've chosen fill_flag 1 (-b/--rgb2 33 33
33)\n");
    printf("chose     your     file     using     -f/--file_name     (-f
name_of_the_file)\n");
    printf("example: ./CW2 --hexagon --start 0 110 --end 110 0 -w 7 --
rgb1 20 20 20 --rgb2 200 100 50 -z 1 --file_name Amogus.bmp\n");
    printf("\n");
    printf("2)to copy & paste, chose option -C/--copy\n");
    printf("chose source file (-f/--file_name) and the file, where you
going to paste to, (-o/--out_file)\n");
    printf("chose upper left coordinates (-s/--start) and lower right (-
e/--end) of the part of source image to be copied\n");
    printf("chose a point on the output file, where the upper left corner
of the copied piece will be (-p/--point)\n");
    printf("example: ./CW2 -C --start 10 200 --end 150 40 -p 30 230 --
file_name file1.bmp --out_file file2.bmp\n");
    printf("\n");
    printf("3)to draw a frame, chose -F/--frame\n");
    printf("select the type of frame using -m/--mode. 0 - simple frame, 1
- Koch line frame, 2 - Minkowski line frame\n");
    printf("If you selected mode 0 chose width of the frame (-w/--width)
and color (-g/--rgb1) only.\n");
    printf("If you selected mode 1 or 2 chose width (-w/--width) of the
frame,\n");
    printf("color of lines of the frame (-g/--rgb1)\n");
    printf("chose to fill(1) or not to fill(0) frame\n");
    printf("select a color, if you've chosen fill_flag 1 (-b/--rgb2)\n");
    printf("chose file -f/--file_name\n");
    printf("example: ./CW2 -F --width 50 -z 1 --rgb1 55 55 55 --rgb2 1 1
1 -f file.bmp\n");
    printf("\n");
    printf("4)to change color of all pixels of the chosen color to the
other color, choose option -G/--changeclr\n");
    printf("chose your file -f/--file_name\n");
    printf("chose color-to-be-changed -g/--rgb1 and finish color -b/--
rgb2\n");
    printf("chose your file -f/--file_name");
    printf("example:  ./CW2  -G  --rgb1  45  45  45  --rgb2  88  88  88  -f
myfile.bmp\n");
    printf("\n");
    printf("5)to see bmp file info chose -i/--info and write your file
name\n");
    printf("example: ./CW2 -i myfile.bmp\n");
```

```
}
int check_cord(int x1, int y1, int x2, int y2){
    int ans = (x1 <= x2) && (y1 >= y2);
    return ans;
}

void printinf(char* s){
    BMP bmp = get_img(s);
    printf("version - %d\n", bmp.version);
    printf("\n");
    printf("Header info:\n%x - signature\n%u - filesize\n%hu -
reserved1\n%hu - reserved2\n%u - pixel offset\n", bmp.head.Type,
bmp.head.Size, bmp.head.Reserved1, bmp.head.Reserved2, bmp.head.OffBits);
    printf("\n");
    printf("Header file info:\n%u - height\n%u - width\n%u - header
size\n%u - bits per pix\n%u - planes\n%x - compression\n",
bmp.inf.Height,    bmp.inf.Width,    bmp.inf.Size,    bmp.inf.BitCount,
bmp.inf.Planes, bmp.inf.Compression);
    printf("%hu - image size\n%u - XPelsPerMeter\n%u - YPelsPerMeter\n%u
-      ClrUsed\n%hu      -      ClrImportant\n",      bmp.inf.SizeImage,
bmp.inf.XPelsPerMeter,      bmp.inf.YPelsPerMeter,      bmp.inf.ClrUsed,
bmp.inf.ClrImportant);
    free_BMP(bmp);
}
```

Название файла: func.c

```
#include "CWlib.h"
#include "func.h"

void change_clr(RGB* source, RGB clr){
    source->r = clr.r;
    source->g = clr.g;
    source->b = clr.b;
}

int cmp_rgb(RGB rgb1, RGB rgb2){
    if(rgb1.r == rgb2.r && rgb1.g == rgb2.g && rgb1.b == rgb2.b){
        return 1;
    }
    return 0;
}

int check_bmp_c_p(BMP sour, BMP paste, int xs1, int ys1, int xs2, int
ys2, int xp, int yp){
    int sour_wid = sour.inf.Width;
    int sour_hei = abs(sour.inf.Height);

    int pa_wid = paste.inf.Width;
    int pa_hei = abs(paste.inf.Height);
    //if x1, y1, x2, y2 inside the pick
    int case1 = (xs1 <0) || (xs1 >= sour_wid);
    int case2 = (ys1 < 0) || (ys1 >= sour_hei);
    int case3 = (xs2 <0) || (xs2 >= sour_wid);
    int case4 = (ys2 < 0) || (ys2 >= sour_hei);
    //if xp, yp
    int case5 = (yp < 0) || (yp >= pa_hei);
    int case6 = (xp < 0) || (xp >= pa_wid);
    //if there is a place
```

```
    int cmp_wid = xs2 - xs1;
    int cmp_hei = ys1 - ys2;

    int case7 = (pa_wid - xp) < cmp_wid;
    int case8 = yp < cmp_hei;
    return (case1 || case2 || case3 || case4 || case5 || case6 || case7
|| case8);
}

void copy_paste(char* source, char* paste_here, int xs1, int ys1, int
xs2, int ys2, int xp, int yp){
    BMP sour = get_img(source);
    BMP paste = get_img(paste_here);
    if(check_bmp_c_p(sour, paste, xs1, ys1, xs2, ys2, xp, yp)){
        printf("impossible coordinates for these pictures!\n");
        exit(1);
    }
    int width = xs2 - xs1;
    int height = ys1 - ys2;
    for(int i = 0; i < height; i++){
        for(int j = 0; j < width; j++){
            change_clr((paste.arr)[yp    -    i]+xp+j,    (sour.arr)[ys1-
i][xs1+j]);
        }
    }
    put_img(paste_here, paste);
    free_BMP(sour);
    free_BMP(paste);
};

void change_color(char* file_name, RGB to_be_changed, RGB base){
    BMP bmp = get_img(file_name);
    for(int i = 0; i < abs(bmp.inf.Height); i++){
        for(int j = 0; j < abs(bmp.inf.Width); j++){
            if(cmp_rgb((bmp.arr)[i][j], to_be_changed)){
                change_clr((bmp.arr)[i]+j, base);
            }
        }
    }
    put_img(file_name, bmp);
    free_BMP(bmp);
}

int lin_func(float b, float c, int y){
    int x = b*y+c;
    return x;
}

void draw_line(BMP bmp, RGB color, int x1, int y1, int x2, int y2){
    if(x1 == x2){
        int min = y1;
        int max = y2;
        if(y1 > y2){
            min = y2;
            max = y1;
        }
        for(int i = min; i <= max; i++){
```

```
            change_clr((bmp.arr)[i]+x1,color);
        }
    }else if (y1 == y2){
        int min = x1;
        int max = x2;
        if(x1 > x2){
            min = x2;
            max = x1;
        }
        for(int i = min; i <= max; i++){
            change_clr((bmp.arr)[y1]+i,color);
        }
    }else if(abs(x1 - x2) > abs(y1-y2)){
        float d_x = x1- x2;;
        float b = (y1-y2)/d_x;
        float c = (x1*y2-y1*x2)/d_x;

        int min = x1;
        int max = x2;
        if(x1 > x2){
            min = x2;
            max = x1;
        }
        int tmp_y;
        for(int i = min; i <= max; i++){
            tmp_y = lin_func(b, c, i);
            change_clr((bmp.arr)[tmp_y]+i, color);
        }
    }else{
        float d_y = y1-y2;
        float b = (x1-x2)/d_y;
        float c = (y1*x2 - x1*y2)/d_y;
        int tmp_x;

        int min = y1;
        int max = y2;
        if(y1 > y2){
            min = y2;
            max = y1;
        }
        for(int i = min; i <= max; i++){
            tmp_x = lin_func(b, c, i);
            change_clr((bmp.arr)[i]+tmp_x,color);
        }
    }
}

void draw_thick_line(BMP bmp, RGB color, int x1, int y1, int x2, int y2,
int width, int flag){ //flaf - 1 - to right 0 - to left or 1 - down 0 -
up
    if(y1 == y2){
        if(flag){
            for(int i = 0; i < width; i++){
                draw_line(bmp, color, x1, y1-i, x2, y2-i);
            }
        }else{
            for(int i = 0; i < width; i++){
```

```
                draw_line(bmp, color, x1, y1+i, x2, y2+i);
            }
        }
    }else {
        int wid = width/0.87; // 0.87 ~ sin(60), but it would be better
to use math.h here, but gcc can't eat it correctly
        if (flag) {
            for (int i = 0; i < wid; i++) {
                draw_line(bmp, color, x1 + i, y1, x2 + i, y2);
            }
        } else {
            for (int i = 0; i < wid; i++) {
                draw_line(bmp, color, x1 - i, y1, x2 - i, y2);
            }
        }
    }
}

void fill_hexagon_v1(RGB** arr, RGB fill_color, int x1, int x2, int y1,
int y2, float* c, float* b){
    for(int x = x1; x <= x2; x++){
        for(int y = y1; y <= y2; y++){
            if(x >= lin_func(b[0], c[0], y) && x <= lin_func(b[1], c[1],
y) && x <= lin_func(b[2], c[2], y) && x >= lin_func(b[3], c[3], y)){
                change_clr(arr[y]+x, fill_color);
            }
        }
    }
}

float b_k(int x1, int y1, int x2, int y2){
    float d_y = y1-y2;
    float b = (x1-x2)/d_y;
    return b;
}

float c_k(int x1, int y1, int x2, int y2){
    float d_y = y1-y2;
    float c = (y1*x2 - y2*x1)/d_y;
    return c;
}

int check_coord_hex(BMP bmp,int x1, int x2, int y1, int y2, int width){
    int hei = abs(bmp.inf.Height);
    int wid = bmp.inf.Width;
    int case1 = x1 < 0 || x1 >= wid;
    int case2 = x2 <0 || x2 >= wid;
    int case3 = y1 < 0 || y1 >= hei;
    int case4 = y2 < 0 || y2 >= hei;
    int case5 = (width > (x2 - x1)) || (width > (y2 - y1));
    return (case1 || case2 || case3 || case4 || case5);
}

void draw_hexagon(char* file_name, int* data, int mode, int width, RGB
line_color, int fill_flag, RGB fill_color){ //mode - 0 - 2
coordinates[x1][y1][x2][y2], 1 - radius and point[rad][x][y]; data -
array with cord/rad&point;
```

```
    int y1, y2, y3, x1, x2, x3, x4; //try to image hexagon and you will
understand why
    int rad;
    if(mode){ //rad&point
        rad = data[0]; int x = data[1]; int y = data[2];
        y2 = y;
        x1 = x - rad;
        x4 = x + rad;
    }else{//two cord
        int x_right = data[0]; int y_up = data[1]; int x_left = data[2];
int y_down = data[3];
        y2 = (y_up+y_down)/2; //y_down + (y_up-y_down)/2   - if you
scared/gonna work with big pictures(height/width more than 2**15)
        x1 = x_right;
        x4 = x_left;
        rad = (x4-x1)/2;
    }
    x2 = x1 + rad/2;
    x3 = x4 - rad/2;
    y3 = y2 + rad*0.866;
    y1 = y2 - rad*0.866;
    BMP bmp = get_img(file_name);

    if(check_coord_hex(bmp, x1, x4, y1, y3, width)){
        printf("impossible coordinates\n");
        exit(1);
    }

    if(fill_flag){
        float* c = malloc(sizeof(float)*4);
        float* b = malloc(sizeof(float)*4);
        if(c == NULL || b == NULL){
            printf("couldn't do malloc in draw_hexagon.\n");
            exit(1);
        }
        c[0] = c_k(x1, y2, x2, y3);
        c[1] = c_k(x3, y3, x4, y2);
        c[2] = c_k(x4, y2, x3, y1);
        c[3] = c_k(x2, y1, x1, y2);

        b[0] = b_k(x1, y2, x2, y3);
        b[1] = b_k(x3, y3, x4, y2);
        b[2] = b_k(x4, y2, x3, y1);
        b[3] =  b_k(x2, y1, x1, y2);
        fill_hexagon_v1(bmp.arr, fill_color, x1, x4, y1, y3, c, b);
        free(c);
        free(b);
    }

    draw_thick_line(bmp, line_color, x1, y2, x2, y3, width, 1); //c[0],
b[0] - left up side
    draw_thick_line(bmp, line_color, x2, y3, x3, y3, width, 1);
    draw_thick_line(bmp, line_color, x3, y3, x4, y2, width, 0); //c[1]
b[1] - right up side
    draw_thick_line(bmp, line_color, x4, y2, x3, y1, width, 0); //c[2]
b[2] - right down side
    draw_thick_line(bmp, line_color, x3, y1, x2, y1, width, 0);
```

```
    draw_thick_line(bmp, line_color, x2, y1, x1, y2, width, 1); //c[3]
b[3] - left down side

    put_img(file_name, bmp);
    free_BMP(bmp);
}
```

Название файла: frame.c

```
#include "CWlib.h"
#include "func.h"
#include <math.h>

//fill algorithm
typedef struct NODE{
    int x;
    int y;
    struct NODE* next;
}NODE;

typedef struct que{
    NODE* tail;
    NODE* head;
    int size;
}que;

NODE* cr_node(int x_, int y_){
    NODE* ans = malloc(sizeof(NODE));
    ans->x = x_;
    ans->y = y_;
    ans->next = NULL;
    return ans;
}

void add_node(que* ochrd, int x, int y){
    NODE* tmp = cr_node(x, y);
    if(ochrd->size){
        ochrd->tail->next = tmp;
        ochrd->tail = tmp;
        ochrd->size++;
    }else {
        ochrd->size = 1;
        ochrd->tail = tmp;
        ochrd->head = tmp;
    }
}

void pop_first_node(que* ochrd){
    NODE* tmp = ochrd->head;
    ochrd->head = ochrd->head->next;
    ochrd->size--;
    free(tmp);
}

void fill_v2(BMP bmp, RGB fill_color, RGB line_color, int a, int b){
    que* ochrd = malloc(sizeof(que));
    ochrd->size = 0;
    add_node(ochrd, a, b);
    int x, y;
```

```
    int case1, case2;
    while(ochrd->size){
        x = ochrd->head->x;
        y = ochrd->head->y;
        case1 = x >= 0 && x < bmp.inf.Width;
        case2 =  y>=0 && y < abs(bmp.inf.Height);
        if(case1 && case2){
            case1 = !cmp_rgb(bmp.arr[y][x], fill_color);
            case2 = !cmp_rgb(bmp.arr[y][x], line_color);
            if(case1 && case2){
                change_clr(bmp.arr[y]+x, fill_color);
                add_node(ochrd, x+1, y);
                add_node(ochrd, x-1, y);
                add_node(ochrd, x, y+1);
                add_node(ochrd, x, y-1);
            }
        }
        pop_first_node(ochrd);
    }
    free(ochrd);
}

//end fill algorithm
//Koch start

int check_bmp(BMP bmp, int width){
    int case1 = width >= (abs(bmp.inf.Height)/2);
    int case2 = width >= bmp.inf.Width/2;
    return (case1 || case2);
}
void draw_Koch_snowflake(BMP bmp, int x1, int y1, int x2, int y2, RGB
color, int n){
    if(n){
        float L = hypot((x1-x2), (y1-y2));
        float cos_x = (x2-x1)/L;
        float sin_x = (y2-y1)/L;
        float h = (L/3) * (sqrt(3)/2);
        int x_new = ((x1+x2)/2)- h*sin_x;
        int y_new = ((y1+y2)/2) + h*cos_x;
        int d_x = x2-x1; int d_y = y2-y1;
        int x_new_1 = x1 +d_x/3; int x_new_2 = x2 - d_x/3;
        int y_new_1 = y1 +d_y/3; int y_new_2 = y2 - d_y/3;

        draw_Koch_snowflake(bmp, x1, y1, x_new_1, y_new_1, color, n-1 );
        draw_Koch_snowflake(bmp, x_new_1, y_new_1, x_new, y_new, color,
n-1 );
        draw_Koch_snowflake(bmp, x_new, y_new, x_new_2, y_new_2, color,
n-1 );
        draw_Koch_snowflake(bmp, x_new_2, y_new_2, x2, y2, color, n-1 );

    }else{
        draw_line(bmp, color, x1, y1, x2, y2);
    }
}

int ret_n(int L_teory){
    int n = 1;
```

```
    if(L_teory > 50) n = 2;
    if(L_teory >= 100) n = 3;
    if(L_teory >= 250) n = 4;
    if(L_teory >= 500) n = 5;
    if(L_teory >= 1000) n = 6;
    return n;
}

void draw_Koch_frame(char* file_name, int width, int fill_flag, RGB
line_color, RGB fill_color){
    BMP bmp_ans = get_img(file_name);
    if(check_bmp(bmp_ans, width)){
        printf("this frame will cover the whole picture! Dont do
that\n");
        exit(1);
    }
    float L_theory = width * 2 * sqrt(3);

    int frame_width = bmp_ans.inf.Width - (2 * width);
    int frame_height = abs(bmp_ans.inf.Height) - (2 * width);

    int actual_num_wid = 1 + frame_width/L_theory;
    int actual_num_hei = 1 + frame_height/L_theory;

    int d_wid = frame_width/actual_num_wid;
    int d_hei = frame_height/actual_num_hei;

    int n_wid = ret_n(d_wid);
    int n_hei = ret_n(d_hei);
    int i;
    for(i = 0; i < actual_num_wid; i++){
        draw_Koch_snowflake(bmp_ans, width + (i+1)*d_wid, width, width +
i*d_wid, width ,line_color , n_wid);
        draw_Koch_snowflake(bmp_ans,        width        +        i*d_wid,
abs(bmp_ans.inf.Height)     -     width,     width     +     (i+1)*d_wid,
abs(bmp_ans.inf.Height) - width , line_color, n_wid);
    }
    draw_line(bmp_ans,    line_color,    width    +    i*d_wid,    width,
bmp_ans.inf.Width - width, width);
    draw_line(bmp_ans,    line_color,    bmp_ans.inf.Width    -    width,
abs(bmp_ans.inf.Height) - width, width + i*d_wid, abs(bmp_ans.inf.Height)
- width);

    for(i = 0; i < actual_num_hei; i++){
        draw_Koch_snowflake(bmp_ans, width, width + i*d_hei, width, width
+ (i+1)*d_hei ,line_color , n_hei);
        draw_Koch_snowflake(bmp_ans, bmp_ans.inf.Width - width, width +
(i+1)*d_hei,  bmp_ans.inf.Width  -  width,  width+  d_hei*i  ,  line_color,
n_hei);
    }
    draw_line(bmp_ans,  line_color,  width,  abs(bmp_ans.inf.Height)  -
width, width, width + i*d_hei);
    draw_line(bmp_ans, line_color, bmp_ans.inf.Width - width, width +
i*d_hei, bmp_ans.inf.Width - width, abs(bmp_ans.inf.Height) - width);

    if(fill_flag){
        fill_v2(bmp_ans, fill_color, line_color, 0, 0);
```

```
    }

    put_img(file_name, bmp_ans);
    free_BMP(bmp_ans);
}
//Koch end
//Minkowski start
void draw_Minkowski_sausage(BMP bmp, int x1, int y1, int x2, int y2, RGB
color, int n){
    if(n){
        if(y1 == y2){
            int x_1 = (3*x1+x2)/4;
            int x_2 = (x1+x2)/2;
            int x_3 = (3*x2+x1)/4;
            int d_y = (x2-x1)/4;

            draw_Minkowski_sausage(bmp, x1, y1, x_1, y1,color, n-1);
            draw_Minkowski_sausage(bmp, x_1, y1, x_1, y1+d_y,color, n-1);
            draw_Minkowski_sausage(bmp,  x_1,  y1+d_y,  x_2,  y1+d_y,color,
n-1);
            draw_Minkowski_sausage(bmp, x_2, y1+d_y, x_2, y1,color, n-1);

            draw_Minkowski_sausage(bmp, x_2, y1, x_2, y1-d_y,color, n-1);
            draw_Minkowski_sausage(bmp,  x_2,  y1-d_y,  x_3,  y1-d_y,color,
n-1);
            draw_Minkowski_sausage(bmp, x_3, y1-d_y, x_3, y1,color, n-1);
            draw_Minkowski_sausage(bmp, x_3, y1, x2, y1, color, n-1);
        }else{
            int y_1 = (3*y1+y2)/4;
            int y_2 = (y1+y2)/2;
            int y_3 = (3*y2+y1)/4;
            int d_x = (y2-y1)/4;
            draw_Minkowski_sausage(bmp, x1, y1, x1, y_1,color, n-1);
            draw_Minkowski_sausage(bmp, x1, y_1, x1-d_x, y_1,color, n-1);
            draw_Minkowski_sausage(bmp,  x1-d_x,  y_1,  x1-d_x,  y_2,color,
n-1);
            draw_Minkowski_sausage(bmp, x1-d_x, y_2, x1, y_2,color, n-1);

            draw_Minkowski_sausage(bmp, x1, y_2, x1+d_x, y_2,color, n-1);
            draw_Minkowski_sausage(bmp,  x1+d_x,  y_2,  x1+d_x,  y_3,color,
n-1);
            draw_Minkowski_sausage(bmp, x1+d_x, y_3, x1, y_3,color, n-1);
            draw_Minkowski_sausage(bmp, x1, y_3, x1, y2,color, n-1);
        }
    }else{
        draw_line(bmp, color, x1, y1, x2, y2);
    }
}

int ret_n_Min(int n){
    int ans = 1;
    if(n > 40) ans = 2;
    if(n >= 130) ans = 3;
    if(n >= 350) ans = 4;
    return ans;
}
```

```
void draw_Minkowski_frame(char* file_name, int width, int fill_flag, RGB
line_color, RGB fill_color){
    BMP bmp = get_img(file_name);
    if(check_bmp(bmp, width)){
        printf("this  frame  will  cover  the  whole  picture!  Dont  do
that\n");
        exit(1);
    }
    int x_right = bmp.inf.Width - width;
    int x_left = width;

    int y_up = bmp.inf.Height - width;
    int y_down = width;

    int k_x = (x_right-x_left)/((width-1)*2);
    int k_y = (y_up-y_down)/((width-1)*2);

    int l_x = (x_right-x_left)/k_x;
    int l_y = (y_up-y_down)/k_y;
    int n = ret_n_Min(l_x);

    int i;
    for(i = 0; i < k_x; i++){
        draw_Minkowski_sausage(bmp,  x_left  +  i*l_x,  y_down,  x_left  +
(i+1)*l_x, y_down, line_color, n);
        draw_Minkowski_sausage(bmp, x_right - (i+1)*l_x, y_up, x_right -
i*l_x, y_up, line_color, n);
    }
    draw_line(bmp,  line_color,  x_left  +  (i)*l_x,  y_down,  x_right,
y_down);
    draw_line(bmp, line_color, x_left, y_up, x_right - (i)*l_x, y_up);

    n = ret_n_Min(l_y);
    for(i = 0; i < k_y; i++){
        draw_Minkowski_sausage(bmp, x_left, y_up - i*l_y , x_left, y_up -
(i+1)*l_y, line_color, n);
        draw_Minkowski_sausage(bmp, x_right, y_down + (i+1)*l_y, x_right,
y_down + i*l_y, line_color, n);
    }
    draw_line(bmp, line_color, x_left, y_up - i*l_y, x_left, y_down);
    draw_line(bmp, line_color, x_right, y_down + i*l_y, x_right, y_up);

     if(fill_flag){
         fill_v2(bmp, fill_color, line_color, 0, 0);
     }
    put_img(file_name, bmp);
    free_BMP(bmp);
}
//Minkowski end
//simple start
void draw_simple_frame(char* file_name, int width, RGB fill_color){
    BMP bmp = get_img(file_name);
    if(check_bmp(bmp, width)){
        printf("this  frame  will  cover  the  whole  picture!  Dont  do
that\n");
        exit(1);
    }
```

```
    int wid = bmp.inf.Width;
    int hei = abs(bmp.inf.Height);
    draw_thick_line(bmp, fill_color,0, 0, 0, hei-1, width, 1);
    draw_thick_line(bmp, fill_color,wid-1, 0, wid-1, hei-1, width, 0);
    draw_thick_line(bmp, fill_color,0, hei-1, wid-1, hei-1, width, 1);
    draw_thick_line(bmp, fill_color,0, 0, wid-1, 0, width, 0);

    draw_thick_line(bmp, fill_color,0, 0, 0, hei-1, width, 1);
    draw_thick_line(bmp, fill_color,wid-1, 0, wid-1, hei-1, width, 0);
    draw_thick_line(bmp, fill_color,0, hei-1, wid-1, hei-1, width, 1);
    draw_thick_line(bmp, fill_color,0, 0, wid-1, 0, width, 0);

    put_img(file_name, bmp);
    free_BMP(bmp);
}
//simple end
```

Название файла: CWlib.h

```
#ifndef basic
#define basic
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#endif

#ifndef CW_2NDSEM

#define CW_2NDSEM
typedef uint8_t BYTE;
typedef uint16_t WORD;
typedef uint32_t DWORD;
typedef short SHORT; /*Yeah, height and width can be negative. Usually
bmp image data starts with bottom-left corner(and in this case H & W are
positive),
 but if it starts with upper-left, then height (and width?) will be
negative.*/

#pragma pack(push, 1)

typedef uint32_t FXPT2DOT30;
typedef struct{
    FXPT2DOT30 ciexyzX;
    FXPT2DOT30 ciexyzY;
    FXPT2DOT30 ciexyzZ;
} CIEXYZ;
typedef struct{
    CIEXYZ ciexyzRed;
    CIEXYZ ciexyzGreen;
    CIEXYZ ciexyzBlue;
} CIEXYZTRIPLE;
//idk why microsoft made this so long. I'm not wise and knowledgeable
enough to understand this.

typedef struct tagBITMAPFILEHEADER {
    WORD  Type;
    DWORD Size;
    WORD  Reserved1;
    WORD  Reserved2;
```

```
    DWORD OffBits;
}HEADERFILE;
typedef struct {
    DWORD        Size;
    DWORD        Width;
    DWORD        Height;
    WORD         Planes;
    WORD         BitCount;
    DWORD        Compression;
    DWORD        SizeImage;
    DWORD        XPelsPerMeter;
    DWORD        YPelsPerMeter;
    DWORD        ClrUsed;
    DWORD        ClrImportant;
    //v.3 ends here (and attributes we gonna use too)
    DWORD        bV5RedMask;
    DWORD        bV5GreenMask;
    DWORD        bV5BlueMask;
    DWORD        bV5AlphaMask;
    DWORD        bV5CSType;
    CIEXYZTRIPLE bV5Endpoints;
    DWORD        bV5GammaRed;
    DWORD        bV5GammaGreen;
    DWORD        bV5GammaBlue;
    //v.4 ends here
    DWORD        bV5Intent;
    DWORD        bV5ProfileData;
    DWORD        bV5ProfileSize;
    DWORD        bV5Reserved;
    //v.5 ends here
}HEADERINFO;

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

typedef struct{
    HEADERFILE head;
    HEADERINFO inf;
    int version; //bmp version 0, 3, 4, 5
    RGB** arr;
    int padding_bytes; //padding bytes to be added to(at?) the end of the
row
}BMP;
#pragma pack(pop)
#endif
```

Название файла: func.h

```
#include "CWlib.h"

#ifndef CW_2NDSEM_READ_WRITE_H
#define CW_2NDSEM_READ_WRITE_H
void free_BMP(BMP bmp);
BMP get_img(char* file_name);
void put_img(char* file_name, BMP bmp);
int not_num_check(char* s);
```

```
void printhelp();
int check_cord(int x1, int y1, int x2, int y2);
void printinf(char* s);
#endif

#ifndef CW_2NDSEM_FUNC_H
#define CW_2NDSEM_FUNC_H
void change_clr(RGB* source, RGB clr);
int cmp_rgb(RGB rgb1, RGB rgb2);
void copy_paste(char* source, char* paste_here, int xs1, int ys1, int
xs2, int ys2, int xp, int yp);
void change_color(char* file_name, RGB to_be_changed, RGB base);
int lin_func(float b, float c, int y);
void draw_line(BMP bmp, RGB color, int x1, int y1, int x2, int y2);
void draw_thick_line(BMP bmp, RGB color, int x1, int y1, int x2, int y2,
int width, int flag);
void draw_hexagon(char* file_name, int* data, int mode, int width, RGB
line_color, int fill_flag, RGB fill_color);
float c(int x1, int y1, int x2, int y2);
float b(int x1, int y1, int x2, int y2);
#endif //CW_2NDSEM_FUNC_H

#ifndef CW_2NDSEM_FRAME_H
#define CW_2NDSEM_FRAME_H
void draw_Koch_frame(char* file_name, int width, int fill_flag, RGB
line_color, RGB fill_color);
void draw_Minkowski_frame(char* file_name, int width, int fill_flag, RGB
line_color, RGB fill_color);
void draw_simple_frame(char* file_name, int width, RGB fill_color);
#endif
```

Название файла: Makefile

```
all: CW2

CW2: main.o read_write.o func.o frame.o
     gcc main.o read_write.o func.o frame.o -o CW2 -lm
main.o: main.c CWlib.h read_write.h
     gcc -c main.c
read_write.o: read_write.c func.h CWlib.h
     gcc -c read_write.c
func.o: func.c func.h CWlib.h
     gcc -c func.c
frame.o: frame.c func.h CWlib.h
     gcc -c frame.c
clean:
     rm -rf *.o
```