

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

КУРСОВАЯ РАБОТА
по дисциплине «Программирование»
Тема: Обработка bmp изображений на языке C

Студент гр. 3388

Пьянков М.Ф.

Преподаватель

Заславский М.М.

Санкт-Петербург

2024

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Пьянков М.Ф.

Группа 3388

Тема работы: Обработка bmp изображений на языке C

Исходные данные:

Вариант 6

Программа обязательно должна иметь CLI (опционально дополнительное использование GUI). Более подробно тут: http://se.moevm.info/doku.php/courses:programming:rules_extra_kurs Программа должна реализовывать весь следующий функционал по обработке bmp-файла

Общие сведения


24 бита на цвет, без сжатия, файл может не соответствовать формату BMP, т.е. необходимо проверка на

BMP формат (дополнительно стоит помнить, что версий у формата несколько).

Если файл не соответствует формату BMP или его версии, то программа должна завершиться с соответствующей ошибкой. Обратите внимание на выравнивание; мусорные данные, если их необходимо дописать в файл для выравнивания, должны быть нулями. обратите внимание на порядок записи пикселей. Все поля стандартных BMP заголовков в выходном файле должны иметь те же значения что и во входном (разумеется кроме тех, которые должны быть изменены).

Программа должна иметь следующие функции по обработке изображений:

(1) Рисование отрезка. Флаг для выполнения данной операции: `--line`.

Отрезок определяется: координатами начала. Флаг `--start`, значение задаётся в формате `'x.y'`, где `x` – координата по `x`, `y` – координата по `y` координатами конца. Флаг `--end` (аналогично флагу `--start`) цветом. Флаг `--color` (цвет задаётся строкой `'rrr.ggg.bbb'`, где `rrr/ggg/bbb` – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)  толщиной. Флаг `--thickness`. На вход принимает число больше 0

(2) Инвертировать цвета в заданной окружности. Флаг для выполнения данной операции: `--inverse_circle`. Окружность определяется координатами ее центра и радиусом. 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

(3) Обрезка изображения. Флаг для выполнения данной операции: `--trim`. Требуется обрезать изображение по заданной области. Область 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`

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

Дата выдачи задания: 6.04.2024

Дата сдачи реферата: 20.05.2024

Дата защиты реферата: 22.05.2024

Студент гр. 3388

Пьянков М. Ф.

Преподаватель

Заславский М. М.

АННОТАЦИЯ

Курсовая работа посвящена разработке программного обеспечения для обработки изображений на языке программирования Си. Программа предоставляет пользователю возможность рисовать прямые линии заданной толщины, инвертировать цвета в пределах заданной окружности и обрезать изображение до любой прямоугольной формы. Для удобства использования программа оснащена графическим интерфейсом командной строки (CLI), который позволяет пользователю задавать необходимые параметры для выполнения операций. В процессе разработки были использованы библиотеки `getopt.h`, стандартная библиотека Си и регулярные выражения для обработки команд.

SUMMARY

The course work is devoted to the development of software for image processing in the C programming language. The program provides the user with the ability to draw straight lines of a given thickness, invert colors within a given circle, and crop the image to any rectangular shape. For ease of use, the program is equipped with a graphical command line interface (CLI), which allows the user to set the necessary parameters to perform operations. During the development process, the `getopt.h` libraries, the standard C library and regular expressions were used to process commands.

СОДЕРЖАНИЕ

Задание на курсовую работу.....	2-3
Аннотация.....	5
Введение.....	8
Ход выполнения работы.....	9-12
Считывание изображения.....	9
Функция <code>Rgb** read_bmp(char* file_name, BitmapFileHeader* bmfh, BitmapInfoHeader* bmif)</code>	9
CLI.....	9-11
Функция <code>int main(int argc, char** argv)</code>	10
Структура <code>longopts</code>	10
Функция <code>getopt_long()</code>	10
Функция <code>switch()</code> , обработка флагов.....	10
Функция <code>int extend_command_message(char* command)</code>	11
Функция <code>int check_args(char* optarg, const char* exp, const char* ext_exp)</code>	11
Обработка изображения.....	11-12
Функция <code>set_pixel(Rgb** arr, int x, int y, int* color, int H, int W)</code>	11
Функция <code>void Circle(Rgb** arr, int xc, int yc, int* color, int rad, int H, int W, int x0, int y0, int x1, int y1)</code>	11
Функция <code>void draw_line(Rgb** arr, int H, int W, int x0, int y0, int x1, int y1, int* color, double thickness)</code>	11
Функция <code>void invert_circle(Rgb** arr, int H, int W, int x0, int y0, int r)</code>	11
Функция <code>Rgb** trim(Rgb** arr, BitmapInfoHeader* bmif, BitmapFileHeader* bmfh, int x0, int y0, int x1, int y1)</code>	12
Функция <code>void write_bmp(char* file_name, Rgb** arr, int H, int W, BitmapFileHeader bmfh, BitmapInfoHeader bmif)</code>	12
Заключение.....	13

Список использованной литературы.....	14
Приложение А с примерами работы программы.....	15-17
Приложение Б. Исходный код программы.....	18-30

ВВЕДЕНИЕ

Цель работы:

Написать программу, обрабатывающую изображение формата bmp[3], согласно командам пользователя.

Подробная характеристика запросов пользователя изложена в разделе ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ (стр. 2)

Основные теоретические положения:

Создана структура Rgb представляющая цвет в формате RGB, поля структуры: r, g, b.

Для хранения данных об изображении создаётся двумерный динамический массив структур Rgb типа Rgb**.

Библиотеки: stdio.h, stdlib.h, string.h, regex.h, getopt.h.

Функции: Обработка изображения: malloc(), fwrite(), fread().

Вывод сообщений, ошибок: exit(), puts(), strcat(), strcpy(), malloc().

Обработка команд: switch(), getopt_long(), sscanf(), regcomp(), regexec().

Подробнее см. в ПРИЛОЖЕНИЕ Б

ХОД ВЫПОЛНЕНИЯ РАБОТЫ

Считывание изображения

Чтение изображения начинается со считывания заголовочного файла и информационного заголовочного файла (BitmapFileHeader, BitmapInfoHeader). Для хранения данных об этом блоке были созданы соответствующие структуры с необходимыми полями.

Далее производится считывание таблицы цветов непосредственно в двумерный массив структур Rgb. Следует обратить внимание на такое понятие как выравнивание: объём памяти занимаемый каждой строкой таблицы цветов в bmp файле кратен 4, поэтому если необходимо, то записываются мусорные данные: $arr[H - i - 1] = \text{malloc}(W * \text{sizeof}(\text{Rgb}) + (4 - ((W * \text{sizeof}(\text{Rgb})) \% 4)) \% 4)$. Для подробностей см. ПРИЛОЖЕНИЕ Б.

Функция *Rgb read_bmp(char* file_name, BitmapFileHeader* bmfh, BitmapInfoHeader* bmif)***

Функция начинается с создания файлового указателя FILE *f. Далее считываются заголовочные блоки функцией fread() и таблица цветов записывается в двумерный динамический массив. Запись происходит таким образом, чтобы было удобнее работать с командами в рамках курсовой работы (начало координат в командах - левый верхний угол, условное начало координат в стандартном считывании — левый нижний угол) поэтому строка считывается с таким индексом: $H - i - 1$. Работа функции разобрана подробнее в методичке к курсовой работе[1].

CLI

Обработка команд производится в функции main(). Реализована она через функцию getopt() и switch(), использовалась библиотека getopt.h[2]. Команды, принимающие аргументы проверяются соответствующими функциями, переопределяются значения флагов для каждой команды. После

функции `switch()` происходит обработка флагов и запуск необходимых функций обработки изображения. см. ПРИЛОЖЕНИЕ Б.

Структура `static struct option longopts[]`

В этой структуре записаны команды в формате, необходимом для работы функции `getopt_long()`.

Функция `int getopt_long(argc, argv, opts, longopts, &longidx)`

Функция обрабатывает массив `char** argv`, подаваемые на вход программы через терминал.

Функция `switch (opt)`

Функция проверяет отдельную команду (`opt`).

Команда `-h (--help)`: выводится сообщение определённое как `HELP_MESSAGE`, программа успешно завершается с кодом выхода 0. См. Рисунок 2.

Команда `-f (--info)`: переопределяется флаг `info_flag = 1`. См. ПРИЛОЖЕНИЕ А: Рисунок 3.

Команда `-i (--input)`: переопределяется флаг `file_name_flag = 1`, задаётся имя входного файла.

Команда `-o(--output)`: переопределяется имя выходного файла (изначально оно определено как `NEW_FILE_NAME`).

Команды `-l(--line)`, `-I(--inverse_circle)`, `-T(--trim)`: переопределяются соответствующие флаги `line_flag`, `inverse_circle_flag`, `trim_flag` на значение 1. См. ПРИЛОЖЕНИЕ А: Рисунок 4, Рисунок 5, Рисунок 6 и Рисунок 10, Рисунок 11, Рисунок 12 соответственно.

Команды `-s(--start)`, `-e(--end)`, `-L(--left_up)`, `-R(--right_down)`: проверка аргументов команды функцией `check_coords_arg(optarg)`, функция `sscanf()` считывает значения в соответствующие переменные.

Команда `-c(--color)`: проверка аргументов функцией `check_color_arg(optarg)` и считывание функцией `sscanf()`.

Команды `-t(--thickness)`, `-r(--radius)`: проверка аргумента функцией `check_one_arg(optarg)` и считывание функцией `sscanf()`.

Для каждой командой предусматривающей аргументы идёт их обработка, и вывод соответствующей информации об ошибке (результат проверки функцией `check_args()` - 2) или завершении программы с кодом ошибки `ERROR_CODE` (результат проверки функцией `check_args()` - 0).

Функция `void extend_command_message(char* command)`

Функция создаёт сообщение об ошибке, когда пользователь ошибся и ввёл слишком много аргументов для команды.

Функция `int check_args(char* optarg, const char* exp, const char* ext_exp)`

Функция проверяет аргументы команды с помощью регулярных выражений: функции `regcomp()`, `regexexec()`. Если аргумент полностью правильный, то функция возвращает 1, если аргументов слишком много — то 2, в остальных случаях считается, что пользователь ввёл неверную команду, возвращается 0. Обработка ошибок пользователя: см. ПРИЛОЖЕНИЕ А: Рисунок 7, Рисунок 8, Рисунок 9.

Обработка изображения

Функция `void set_pixel(Rgb** arr, int x, int y, int* color, int H, int W)`

Проверяет вхождение ячейки в таблицу цветов и меняет её цвет на заданный.

Функция `void Circle(Rgb** arr, int xc, int yc, int* color, int rad, int H, int W, int x0, int y0, int x1, int y1)`

Функция использует алгоритм Брезенхема для рисования окружностей.

Функция *void draw_line(Rgb arr, int H, int W, int x0, int y0, int x1, int y1, int* color, double thickness)***

Функция использует алгоритм Брезенхема для рисования линии. Каждый элемент линии представлен окружностей, рисуемой функцией Circle().

Функция *void invert_circle(Rgb arr, int H, int W, int x0, int y0, int r)***

Функция используя двойной цикл и уравнение круга меняет цвет подходящих пикселей (инвертирует) с помощью функции set_pixel().

Функция *Rgb trim(Rgb** arr, BitmapInfoHeader* bmif, BitmapFileHeader* bmfh, int x0, int y0, int x1, int y1)***

Функция используя двойной цикл и уравнение прямоугольника (int rectangle_func(int x, int y, int x0, int y0, int x1, int y1), заполняет новый динамический массив Rgb new_arr, тем самым выполняя обрезку изображения. Функция также переопределяет поля заголовочных блоков в соответствии с заданными параметрами.

void write_bmp(char* file_name, Rgb** arr, int H, int W, BitmapFileHeader bmfh, BitmapInfoHeader bmif)

Функция аналогично функции read_bmp(), записывает данные в файл, также создавая файловый указатель и применяя выравнивание. Разбор функции представлен в методичке к курсовой[1].

Для подробностей см. ПРИЛОЖЕНИЕ Б.

ЗАКЛЮЧЕНИЕ

В результате написания курсовой работы было создано консольное приложение обрабатывающее bmp изображения согласно инструкциям пользователя. Интерфейс представлен интерфейсом командной строки (CLI) Реализован следующий функционал: рисование линии заданной толщиной, инвертирования цветов в заданной окружности и обрезка изображения до любой прямоугольной формы.

Все задачи были успешно выполнены. Предусмотрены случаи ошибочного ввода пользователем и вывод сообщения об ошибке или завершение программы с соответствующим кодом.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

БАЗОВЫЕ СВЕДЕНИЯ К ВЫПОЛНЕНИЮ КУРСОВОЙ РАБОТЫ ПО
ДИСЦИПЛИНЕ «ПРОГРАММИРОВАНИЕ». ВТОРОЙ СЕМЕСТР

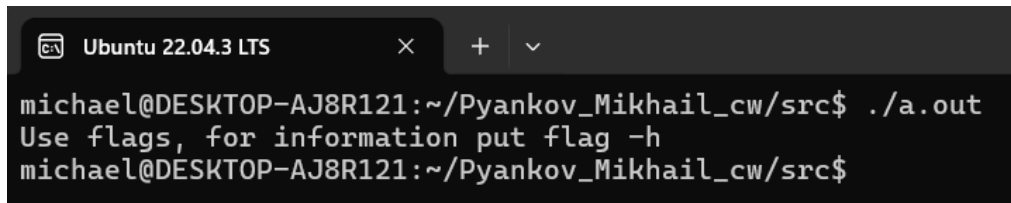
(Методичка)[1]

getopt[2]

ВМР формат[3]

ПРИЛОЖЕНИЕ А. С ПРИМЕРАМИ РАБОТЫ ПРОГРАММЫ

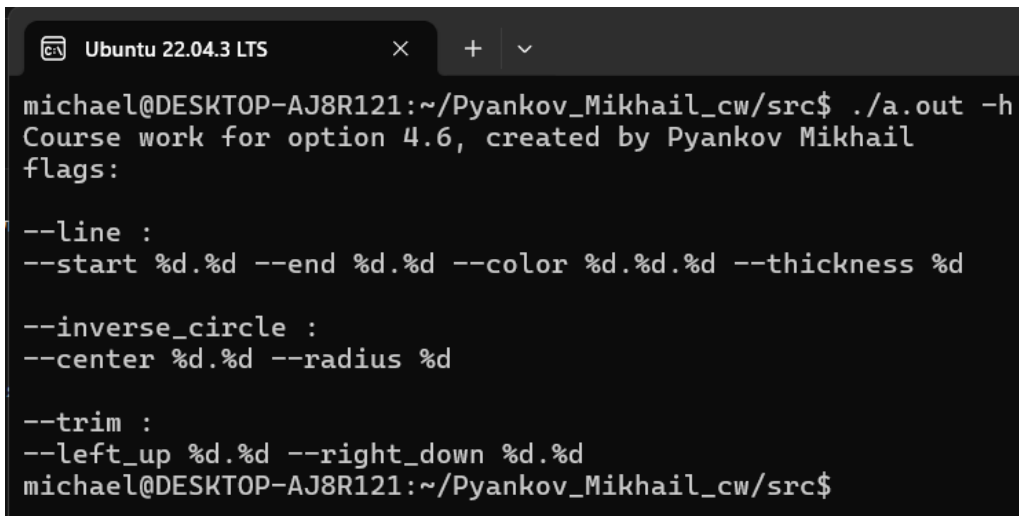
Пользователь не ввёл команд



```
Ubuntu 22.04.3 LTS
michael@DESKTOP-AJ8R121:~/Pyankov_Mikhail_cw/src$ ./a.out
Use flags, for information put flag -h
michael@DESKTOP-AJ8R121:~/Pyankov_Mikhail_cw/src$
```

Рисунок 1 — Запуск без команд

Флаг -h



```
Ubuntu 22.04.3 LTS
michael@DESKTOP-AJ8R121:~/Pyankov_Mikhail_cw/src$ ./a.out -h
Course work for option 4.6, created by Pyankov Mikhail
flags:

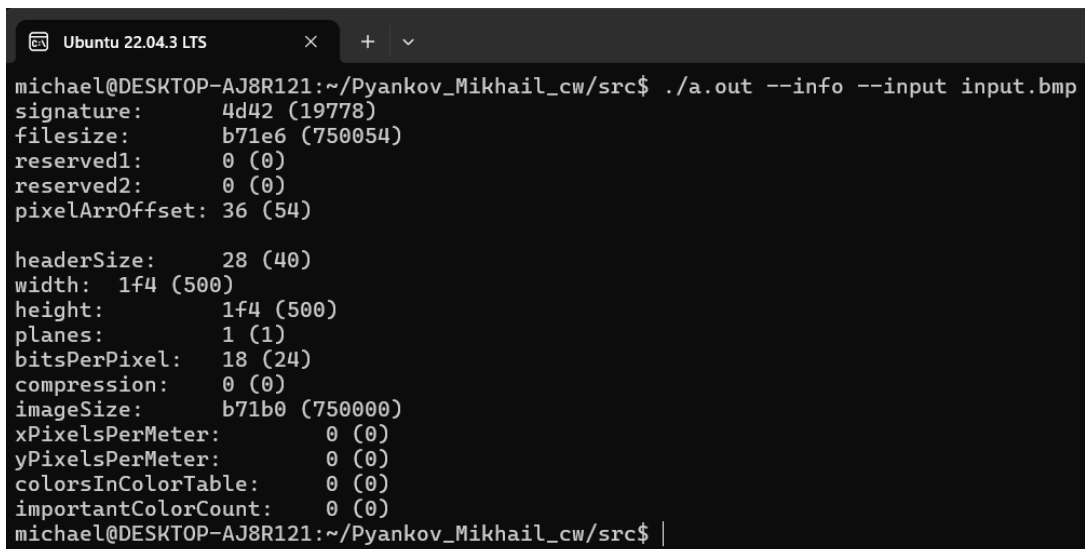
--line :
--start %d.%d --end %d.%d --color %d.%d.%d --thickness %d

--inverse_circle :
--center %d.%d --radius %d

--trim :
--left_up %d.%d --right_down %d.%d
michael@DESKTOP-AJ8R121:~/Pyankov_Mikhail_cw/src$
```

Рисунок 2 — Запуск с командой -h

Флаг --info



```
Ubuntu 22.04.3 LTS
michael@DESKTOP-AJ8R121:~/Pyankov_Mikhail_cw/src$ ./a.out --info --input input.bmp
signature:      4d42 (19778)
filesize:       b71e6 (750054)
reserved1:      0 (0)
reserved2:      0 (0)
pixelArrOffset: 36 (54)

headerSize:     28 (40)
width: 1f4 (500)
height: 1f4 (500)
planes:         1 (1)
bitsPerPixel:   18 (24)
compression:    0 (0)
imageSize:      b71b0 (750000)
xPixelsPerMeter: 0 (0)
yPixelsPerMeter: 0 (0)
colorsInColorTable: 0 (0)
importantColorCount: 0 (0)
michael@DESKTOP-AJ8R121:~/Pyankov_Mikhail_cw/src$ |
```

Рисунок 3 — Запуск с командой --info

Флаг —line

```
Ubuntu 22.04.3 LTS
michael@DESKTOP-AJ8R121:~/Pyankov_Mikhail_cw/src$ ./a.out --line --start 100.200 --end 400.400 --thickness 40 --color 24
4.0.0 --input input.bmp --output output.bmp
michael@DESKTOP-AJ8R121:~/Pyankov_Mikhail_cw/src$
```

Рисунок 10 — Команда рисования линии



Рисунок 4 — Запуск с командой --line

Флаг —inverse_circle

```
Ubuntu 22.04.3 LTS
michael@DESKTOP-AJ8R121:~/Pyankov_Mikhail_cw/src$ ./a.out --inverse_circle --center 250.200 --radius 100 --input input.b
mp --output output.bmp
michael@DESKTOP-AJ8R121:~/Pyankov_Mikhail_cw/src$
```

Рисунок 11 — Команда инвертирования цветов в окружности

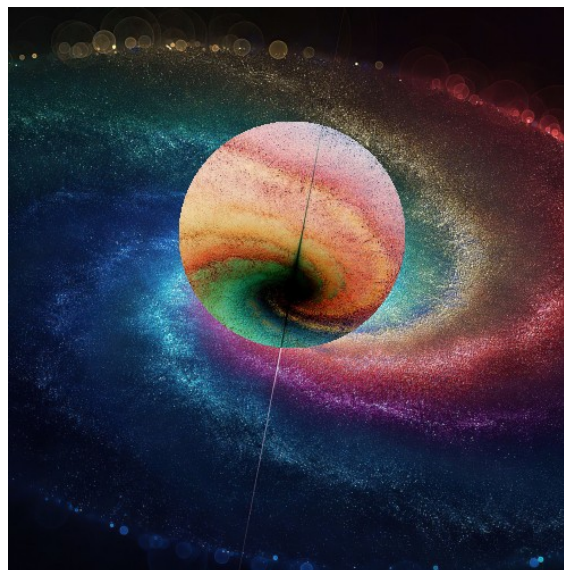


Рисунок 5 — Запуск с командой --inverse_circle

Флаг --trim

```
Ubuntu 22.04.3 LTS
michael@DESKTOP-AJ8R121:~/Pyankov_Mikhail_cw/src$ ./a.out --trim --left_up 0.0 --right_down 400.400 --input input.bmp --output output.bmp
michael@DESKTOP-AJ8R121:~/Pyankov_Mikhail_cw/src$ |
```

Рисунок 12 — Команда обрезки изображения



Рисунок 6 — Запуск с командой --trim

Ошибка в команде

```
Ubuntu 22.04.3 LTS
michael@DESKTOP-AJ8R121:~/Pyankov_Mikhail_cw/src$ ./a.out --inverse_circle --center 250.200 --radius -100 --input input.bmp --output output.bmp
Invalid command!
michael@DESKTOP-AJ8R121:~/Pyankov_Mikhail_cw/src$
```

Рисунок 7 — Ошибка в команде

Слишком много аргументов

```
Ubuntu 22.04.3 LTS
michael@DESKTOP-AJ8R121:~/Pyankov_Mikhail_cw/src$ ./a.out --line --start 100.200 --end 400.400 --thickness 40 --color 24 4.0.0.0 --input input.bmp --output output.bmp
Too much args for --color flag, only first will be write!
michael@DESKTOP-AJ8R121:~/Pyankov_Mikhail_cw/src$ |
```

Рисунок 8 — Запуск с флагом с чрезмерным количеством параметров

Не указан входной файл

```
Ubuntu 22.04.3 LTS
michael@DESKTOP-AJ8R121:~/Pyankov_Mikhail_cw/src$ ./a.out --line --start 100.200 --end 400.500 --thickness 40 --color 24 4.0.0
There is no file with that name!
michael@DESKTOP-AJ8R121:~/Pyankov_Mikhail_cw/src$
```

Рисунок 9 — Запуск программы без входного файла

ПРИЛОЖЕНИЕ Б. ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main_cw.c

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

#define no_argument          0
#define required_argument   1
#define SIZE 10
#define COMMAND_SIZE 50
#define GROUPS 1
#define NEW_FILE_NAME "./out.bmp"
#define HELP_MESSAGE "Course work for option 4.6, created by Pyankov Mikhail\n\nflags:\n\n--line : \n\n--start %d.%d --end %d.%d --color %d.%d.\n%d --thickness %d\n\n--inverse_circle : \n\n--center %d.%d --radius %d\n\n--trim : \n\n--left_up %d.%d --right_down %d.%d"
#define DEFAULT_MESSAGE "Use flags, for information put flag -h"

#define REG_COORDS "^-?[0-9]+\\.?-?[0-9]+$"
#define REG_COORDS_EXTENDED "^-?[0-9]+\\.?-?[0-9]+(\\.?-?[0-9]+)+$"

#define REG_LINE_COLOR "^[0-9]+\\.?[0-9]+\\.?[0-9]+$"
#define REG_LINE_COLOR_EXTENDED "^[0-9]+\\.?[0-9]+\\.?[0-9]+(\\.?-?[0-9]+)+$"

#define REG_ONE_ARG "^[0-9]+$"
#define REG_ONE_ARG_EXTENDED "^[0-9]+(\\.?-?[0-9]+)+$"

#define FILE_NAME "[0-9A-z~@#\\$\\^_\\(\\)\\{\\}\\'`]+\\.?[0-9A-z~@#\\$\\^_\\(\\)\\{\\}\\'`]+$"

#define INVALID_COMMAND "Invalid command!"
#define EXTENDED_COMMAND_1 "Too much args for "
#define EXTENDED_COMMAND_2 " flag, only first will be write!"

#define LINE_COMMAND_ERROR "Missed args for --line flag!"
#define INVERSE_CIRCLE_COMMAND_ERROR "Missed args for --inverse_circle flag!"
#define TRIM_COMMAND_ERROR "Missed args for --trim flag!"
#define TRIM_COORDS_ERROR "--trim coords error!"

#define FILE_NAME_FORGOTTEN "Put file_name!"
#define FILE_NAME_ERROR "There is no file with that name!"
#define FILE_TYPE_ERROR "File corrupted!"
#define OUTPUT_FILE_ERROR "Output file name error, default name will be out.BMP!"
#define SAME_INP_OUT_FILES "Input and output files are same!"
#define ERROR_CODE 40

#pragma pack (push, 1)
```

```

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

typedef struct {
    unsigned int headerSize;
    unsigned int width;
    unsigned int height;
    unsigned short planes;
    unsigned short bitsPerPixel;
    unsigned int compression;
    unsigned int imageSize;
    unsigned int xPixelsPerMeter;
    unsigned int yPixelsPerMeter;
    unsigned int colorsInColorTable;
    unsigned int importantColorCount;
} BitmapInfoHeader;

typedef struct {
    unsigned char b;
    unsigned char g;
    unsigned char r;
} Rgb;

#pragma pack(pop)

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:\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);
    }

void extend_command_message(char* command)
{
    char* message = malloc(COMMAND_SIZE);
    strcpy(message, EXTENDED_COMMAND_1);
    strcat(message, command);
    strcat(message, EXTENDED_COMMAND_2);
    puts(message);
}

int check_args(char* optarg, const char* exp, const char* ext_exp)
{
    regex_t regex_compiled;
    regmatch_t group_array[GROUPS];

    regcomp(&regex_compiled, exp, REG_EXTENDED);
    if(regexec(&regex_compiled, optarg, GROUPS, group_array, 0) == 0)
        return 1;
    regcomp(&regex_compiled, ext_exp, REG_EXTENDED);
    if(regexec(&regex_compiled, optarg, GROUPS, group_array, 0) == 0)
        return 2;
    return 0;
}

int check_bmp(char* file_name)
{
    FILE *f = fopen(file_name, "rb");
    BitmapFileHeader bmfh;
    BitmapInfoHeader bmif;
    fread(&bmfh, 1, sizeof(BitmapFileHeader), f);
    fread(&bmif, 1, sizeof(BitmapInfoHeader), f);
    if((bmfh.signature == 19778 || bmfh.signature == 16706 ||
bmfh.signature == 674115 || bmfh.signature == 675907 || bmfh.signature
== 672585 || bmfh.signature == 676944)
    && bmif.bitsPerPixel == 24 && bmif.compression == 0);
    return 1;
    return 0;
}

int check_file_name(char* file_name)
{
    FILE *f = fopen(file_name, "rb");
    if(f == NULL)
        return 0;

```

```

        return 1;
    }

int check_file_str(char* file_name)
{
    regex_t regex_compiled;
    regmatch_t group_array[GROUPS];

    regcomp(&regex_compiled, FILE_NAME, REG_EXTENDED);
    if(regexec(&regex_compiled, file_name, GROUPS, group_array, 0) ==
0)
        return 1;
    return 0;
}

Rgb** read_bmp(char* file_name, BitmapFileHeader* bmfh,
BitmapInfoHeader* bmif)
{
    FILE *f = fopen(file_name, "rb");
    fread(bmfh, 1, sizeof(BitmapFileHeader), f);
    fread(bmif, 1, sizeof(BitmapInfoHeader), f);
    unsigned int H = bmif->height;
    unsigned int W = bmif->width;
    Rgb** arr = malloc(H * sizeof(Rgb*));
    for(int i = 0; i < H; i++)
    {
        arr[H - i - 1] = malloc(W * sizeof(Rgb) + (4 - ((W *
sizeof(Rgb)) % 4) ) % 4);
        fread(arr[H - i - 1], 1, W * sizeof(Rgb) + (4 - ((W *
sizeof(Rgb)) % 4) ) % 4, f);
    }
    fclose(f);
    return arr;
}

int circle_func(int x, int y, int x0, int y0, int r)
{
    return ((x - x0) * (x - x0) + (y - y0) * (y - y0)) < (r*r);
}

void set_pixel(Rgb** arr, int x, int y, int* color, int H, int W)
{
    if(y >= H || y < 0 || x >= W || x < 0)
        return;
    arr[y][x].r = *(color);
    arr[y][x].g = *(color + 1);
    arr[y][x].b = *(color + 2);
}

void Circle(Rgb** arr, int X1, int Y1, int* color, int R, int H, int
W)
{
    int x = 0;
    int y = R;
    int delta = 1 - 2 * R;

```

```

int error = 0;
while (y >= x)
{
    set_pixel(arr, X1 + x, Y1 + y, color, H, W);
    set_pixel(arr, X1 + x, Y1 - y, color, H, W);
    set_pixel(arr, X1 - x, Y1 + y, color, H, W);
    set_pixel(arr, X1 - x, Y1 - y, color, H, W);
    set_pixel(arr, X1 + y, Y1 + x, color, H, W);
    set_pixel(arr, X1 + y, Y1 - x, color, H, W);
    set_pixel(arr, X1 - y, Y1 + x, color, H, W);
    set_pixel(arr, X1 - y, Y1 - x, color, H, W);
    error = 2 * (delta + y) - 1;
    if((delta < 0) && (error <= 0))
    {
        delta += 2 * ++x + 1;
        continue;
    }

    if((delta > 0) && (error > 0))
    {
        delta -= 2 * --y + 1;
        continue;
    }
    delta += 2 * (++x - --y);
}
}

void draw_line(Rgb** arr, int H, int W, int x0, int y0, int x1, int
y1, int* color, int thickness)
{
    int dx = (x1 - x0 >= 0 ? 1 : -1);
    int dy = (y1 - y0 >= 0 ? 1 : -1);
    int lengthX = abs(x1 - x0);
    int lengthY = abs(y1 - y0);
    int length = (lengthX - lengthY > 0 ? lengthX : lengthY);
    if (length == 0)
        Circle(arr, x0, y0, color, thickness, H, W);
    int x = x0;
    int y = y0;
    if(lengthY <= lengthX)
    {
        int d = -lengthX;
        length++;
        while (length--)
        {
            if(x >= 0 && y >= 0 && x < W && y < H);
            Circle(arr, x, y, color, thickness, H, W);
            x += dx;
            d += 2 * lengthY;
            if(d > 0)
            {
                d -= 2 * lengthX;
                y += dy;
            }
        }
    }
}

```

```

    }
    else
    {
        int d = -lengthY;
        length++;
        while (length-->0)
        {
            if(x >= 0 && y >= 0 && x < W && y < H);
            Circle(arr, x, y, color, thickness, H, W);
            y += dy;
            d += 2 * lengthX;
            if(d > 0)
            {
                d -= 2 * lengthY;
                x += dx;
            }
        }
    }
}

void invert_circle(Rgb** arr, int H, int W, int x0, int y0, int r)
{
    for(int i = 0; i < H; i++)
    {
        for(int j = 0; j < W; j++)
        {
            if(circle_func(j, i, x0, y0, r))
            {
                int color[3] = {255 - arr[i][j].r, 255 - arr[i][j].g,
255 - arr[i][j].b};
                set_pixel(arr, j, i, color, H, W);
            }
        }
    }
}

int rectangle_func(int x, int y, int x0, int y0, int x1, int y1)
{
    return (x >= x0 && y >= y0) && (x <= x1 && y <= y1);
}

Rgb** trim(Rgb** arr, BitmapInfoHeader* bmif, BitmapFileHeader* bmfh,
int x0, int y0, int x1, int y1)
{
    int H = bmif->height;
    int W = bmif->width;

    if(x0 >= x1 || y0 >= y1 || x1 < 0 || y1 < 0 || y0 > H || x0 > W)
    {
        puts(TRIM_COORDS_ERROR);
        exit(ERROR_CODE);
    }

    x0 = x0 >= 0 ? x0 : 0;
    y0 = y0 >= 0 ? y0 : 0;

```

```

    y1 = y1 < H ? y1 : H;
    x1 = x1 < W ? x1 : W;
    int new_W = x1 - x0;
    int new_H = y1 - y0;
    Rgb** new_arr = malloc(new_H * sizeof(Rgb *));
    for (int i = 0; i < new_H; i++)
    {
        new_arr[i] = malloc(new_W * sizeof(Rgb) + (4 - ((new_W *
sizeof(Rgb)) %4) ) % 4);
    }
    for(int i = 0; i < H; i++)
    {
        for(int j = 0; j < W; j++)
        {
            if(rectangle_func(j, i, x0, y0, x1, y1))
            {
                int color[3] = {arr[i][j].r, arr[i][j].g, arr[i]
[j].b};
                set_pixel(new_arr, j - x0, i - y0, color, new_H,
new_W);
            }
        }
        bmif->height = new_H;
        bmif->width = new_W;
        bmfh->filesize = (new_H)*((new_W)*sizeof(Rgb) + (4 - ((new_W *
sizeof(Rgb)) %4) ) % 4);
        bmif->imageSize = new_H*new_W;
        return new_arr;
    }

void write_bmp(char* file_name, Rgb** arr, int H, int W,
BitmapFileHeader bmfh, BitmapInfoHeader bmif)
{
    FILE *ff = fopen(file_name, "wb");
    fwrite(&bmfh, 1, sizeof(BitmapFileHeader), ff);
    fwrite(&bmif, 1, sizeof(BitmapInfoHeader), ff);
    for(int i = 0; i < H; i++)
    {
        fwrite(arr[H - i - 1], 1, W * sizeof(Rgb) + (4 - ((W *
sizeof(Rgb)) %4) ) % 4, ff);
    }
    fclose(ff);
}

void romb_draw_line(Rgb** arr, int H, int W, int x0, int y0, int x1,
int y1, int* color, int thickness)
{
    draw_line(arr, H, W, x0, y0, x1, y1, color, (thickness-1)/2);
    if(thickness % 2 == 0)
    {
        if(fabs((float)(x0 - x1)) <= fabs((float)(y0 - y1)))
            draw_line(arr, H, W, x0+1, y0, x1+1, y1, color, (thickness-
1)/2);
        else

```



```

        draw_line(arr, H, W, x0, y0+1, x1, y1+1, color, (thickness-
1)/2);
    }
}

int fill_romb(Rgb** arr, int H, int W, int xv, int yv, int* color,
int** used)
{
    if(xv < H && xv >= 0 && yv < W && yv >=0 && !used[yv][xv] && !
(arr[yv][xv].r == color[0] && arr[yv][xv].g == color[1] && arr[yv]
[xv].b == color[2]))
    {
        set_pixel(arr, xv, yv, color, H, W);
        used[yv][xv] = 1;
        return fill_romb(arr, H, W, xv+1, yv, color, used) ||
fill_romb(arr, H, W, xv-1, yv, color, used) ||
        fill_romb(arr, H, W, xv, yv+1, color, used) || fill_romb(arr,
H, W, xv, yv-1, color, used);
    }
    else
        return 0;
}

void romb(Rgb** arr, int H, int W, int xv, int yv, int size, int*
color)
{
    int thickness = 1;
    double sqrt2 = 1.41421356237;

    double x0 = xv;
    double y0 = yv;

    double x1 = xv-size/sqrt2;
    double y1 = yv+size/sqrt2;

    double x2 = xv;
    double y2 = yv+2*size/sqrt2;

    double x3 = xv+size/sqrt2;
    double y3 = yv+size/sqrt2;

    romb_draw_line(arr, H, W, x0, y0, x1, y1, color, (thickness-1)/2);
    romb_draw_line(arr, H, W, x1, y1, x2, y2, color, (thickness-1)/2);
    romb_draw_line(arr, H, W, x2, y2, x3, y3, color, (thickness-1)/2);
    romb_draw_line(arr, H, W, x3, y3, x0, y0, color, (thickness-1)/2);

    int** used = malloc(sizeof(int*) * H);
    for(int i = 0; i < H; i++)
    {
        used[i] = malloc(sizeof(int) * W);
    }

    for(int i = 0; i < H; i++)
    {

```

```

        for(int j = 0; j < W; j++)
            used[i][j] = 0;
    }

    fill_romb(arr, H, W, xv, yv+2, color, used);
    fill_romb(arr, H, W, xv, yv+size/sqrt2, color, used);
    fill_romb(arr, H, W, xv, yv+10, color, used);
}

int main(int argc, char** argv)
{
    char* file_name;
    int file_name_flag = 0;
    char* new_file_name = NEW_FILE_NAME;

    BitmapFileHeader bmfh;
    BitmapInfoHeader bmif;
    Rgb** array;

    char *opts = "hls:e:c:t:IC:r:TL:R:o:i:f";
    int longidx;
    static struct option longopts[] =
    {
        {"help", no_argument, NULL, 'h'},
        {"line", no_argument, NULL, 'l'},
        {"start", required_argument, NULL, 's'},
        {"end", required_argument, NULL, 'e'},
        {"color", required_argument, NULL, 'c'},
        {"thickness", required_argument, NULL, 't'},
        {"inverse_circle", no_argument, NULL, 'I'},
        {"center", required_argument, NULL, 'C'},
        {"radius", required_argument, NULL, 'r'},
        {"trim", no_argument, NULL, 'T'},
        {"left_up", required_argument, NULL, 'L'},
        {"right_down", required_argument, NULL, 'R'},
        {"info", no_argument, NULL, 'f'},
        {"output", required_argument, NULL, 'o'},
        {"input", required_argument, NULL, 'i'},
        {"square_rhombus", no_argument, NULL, 'S'},
        {"upper_vertex", required_argument, NULL, 'u'},
        {"size", required_argument, NULL, 'z'},
        {"fill_color", required_argument, NULL, 'F'},
        {NULL, no_argument, NULL, 0 }
    };

    int opt = getopt_long(argc, argv, opts, longopts, &longidx);

    if(argc == 1)
    {
        puts(DEFAULT_MESSAGE);
        exit(0);
    }

    int cnt_commands = 0;

```

```

int info_flag = 0;

int x0,y0,x1,y1,thickness;
int color[3];
    int line_flag = 0; int start_flag = 0; int end_flag = 0; int
thickness_flag = 0; int color_flag = 0;

int xc,yc,rad;
    int center_flag = 0; int rad_flag = 0; int inverse_circle_flag =
0;

int xl,yl,xr,yr;
int left_flag = 0; int right_flag = 0; int trim_flag = 0;

int rombus_flag = 0;
int vert_flag = 0;
int size_flag = 0;
int fill_flag = 0;
int fill_color[3];
int rb_size;
int xv, yv;

while(opt != -1)
{
    switch (opt)
    {
        case('h'):
        {
            puts(HELP_MESSAGE);
            exit(0);
        }
        case('f'):
        {
            info_flag = 1;
            cnt_commands++;
            break;
        }
        case('i'):
        {
            file_name_flag = 1;
            file_name = optarg;
            cnt_commands+=2;
            break;
        }
        case('o'):
        {
            cnt_commands+=2;
            if(check_file_str(optarg))
            {
                new_file_name = optarg;
            }
            else
            puts(OUTPUT_FILE_ERROR);
            break;
        }
    }
}

```

```

        case('l'):
        {
            line_flag = 1;
            cnt_commands++;
            break;
        }
        case('s'):
        {
            if(check_args(optarg, REG_COORDS,
REG_COORDS_EXTENDED))
                sscanf(optarg, "%d.%d", &x0, &y0);
            if(check_args(optarg, REG_COORDS, REG_COORDS_EXTENDED)
== 2)
                extend_command_message("--start");
            if(!check_args(optarg, REG_COORDS,
REG_COORDS_EXTENDED))
            {
                puts(INVALID_COMMAND);
                exit(ERROR_CODE);
            }
            start_flag = 1;
            cnt_commands+=2;
            break;
        }
        case('e'):
        {
            if(check_args(optarg, REG_COORDS,
REG_COORDS_EXTENDED))
                sscanf(optarg, "%d.%d", &x1, &y1);
            if(check_args(optarg, REG_COORDS, REG_COORDS_EXTENDED)
== 2)
                extend_command_message("--end");
            if(!check_args(optarg, REG_COORDS,
REG_COORDS_EXTENDED))
            {
                puts(INVALID_COMMAND);
                exit(ERROR_CODE);
            }
            end_flag = 1;
            cnt_commands+=2;
            break;
        }
        case('c'):
        {
            if(check_args(optarg, REG_LINE_COLOR,
REG_LINE_COLOR_EXTENDED))
                sscanf(optarg, "%d.%d.%d", &color[0], &color[1],
&color[2]);
            if(check_args(optarg, REG_LINE_COLOR,
REG_LINE_COLOR_EXTENDED) == 2)
                extend_command_message("--color");
            if(!check_args(optarg, REG_LINE_COLOR,
REG_LINE_COLOR_EXTENDED) || (color[0] > 255 || color[1] > 255 ||
color[2] > 255))
            {

```

```

        puts(INVALID_COMMAND);
        exit(ERROR_CODE);
    }
    color_flag = 1;
    cnt_commands+=2;
    break;
}
case('t'):
{
    if(check_args(optarg, REG_ONE_ARG,
REG_ONE_ARG_EXTENDED))
        sscanf(optarg, "%d", &thickness);
    if(check_args(optarg, REG_ONE_ARG,
REG_ONE_ARG_EXTENDED) == 2)
        extend_command_message("--thickness");
    if(!check_args(optarg, REG_ONE_ARG,
REG_ONE_ARG_EXTENDED))
    {
        puts(INVALID_COMMAND);
        exit(ERROR_CODE);
    }
    thickness_flag = 1;
    cnt_commands+=2;
    break;
}
case('I'):
{
    inverse_circle_flag = 1;
    cnt_commands++;
    break;
}
case('C'):
{
    if(check_args(optarg, REG_COORDS,
REG_COORDS_EXTENDED))
        sscanf(optarg, "%d.%d", &xc, &yc);
    if(check_args(optarg, REG_COORDS, REG_COORDS_EXTENDED)
== 2)
        extend_command_message("--center");
    if(!check_args(optarg, REG_COORDS,
REG_COORDS_EXTENDED))
    {
        puts(INVALID_COMMAND);
        exit(ERROR_CODE);
    }
    center_flag = 1;
    cnt_commands+=2;
    break;
}
case('r'):
{
    if(check_args(optarg, REG_ONE_ARG,
REG_ONE_ARG_EXTENDED))
        sscanf(optarg, "%d", &rad);

```

```

                                if (check_args (optarg,  REG_ONE_ARG,
REG_ONE_ARG_EXTENDED) == 2)
                                extend_command_message ("--radius");
                                if (!check_args (optarg,  REG_ONE_ARG,
REG_ONE_ARG_EXTENDED))
                                {
                                    puts (INVALID_COMMAND);
                                    exit (ERROR_CODE);
                                }
                                rad_flag = 1;
                                cnt_commands+=2;
                                break;
                            }
                            case ('T'):
                            {
                                trim_flag = 1;
                                cnt_commands++;
                                break;
                            }
                            case ('L'):
                            {
                                if (check_args (optarg,  REG_COORDS,
REG_COORDS_EXTENDED))
                                sscanf (optarg, "%d.%d", &xl, &yl);
                                if (check_args (optarg,  REG_COORDS,  REG_COORDS_EXTENDED)
== 2)
                                extend_command_message ("--left_up");
                                if (!check_args (optarg,  REG_COORDS,
REG_COORDS_EXTENDED))
                                {
                                    puts (INVALID_COMMAND);
                                    exit (ERROR_CODE);
                                }
                                left_flag = 1;
                                cnt_commands+=2;
                                break;
                            }
                            case ('R'):
                            {
                                if (check_args (optarg,  REG_COORDS,
REG_COORDS_EXTENDED))
                                sscanf (optarg, "%d.%d", &xr, &yr);
                                if (check_args (optarg,  REG_COORDS,  REG_COORDS_EXTENDED)
== 2)
                                extend_command_message ("--right_down");
                                if (!check_args (optarg,  REG_COORDS,
REG_COORDS_EXTENDED))
                                {
                                    puts (INVALID_COMMAND);
                                    exit (ERROR_CODE);
                                }
                                right_flag = 1;
                                cnt_commands+=2;
                                break;
                            }
                        }

```

```

        case('S'):
        {
            rombus_flag = 1;
            break;
            cnt_commands++;
        }
        case('u'):
        {
            if(check_args(optarg, REG_COORDS,
REG_COORDS_EXTENDED))
                sscanf(optarg, "%d.%d", &xv, &yv);
            if(check_args(optarg, REG_COORDS, REG_COORDS_EXTENDED)
== 2)
                extend_command_message("--upper_vertex");
            if(!check_args(optarg, REG_COORDS,
REG_COORDS_EXTENDED))
            {
                puts(INVALID_COMMAND);
                exit(ERROR_CODE);
            }
            cnt_commands+=2;
            vert_flag = 1;
            break;
        }
        case('z'):
        {
            if(check_args(optarg, REG_ONE_ARG,
REG_ONE_ARG_EXTENDED))
                sscanf(optarg, "%d", &rb_size);
            if(check_args(optarg, REG_ONE_ARG,
REG_ONE_ARG_EXTENDED) == 2)
                extend_command_message("--size");
            if(!check_args(optarg, REG_ONE_ARG,
REG_ONE_ARG_EXTENDED))
            {
                puts(INVALID_COMMAND);
                exit(ERROR_CODE);
            }
            size_flag = 1;
            cnt_commands+=2;
            break;
        }
        case('F'):
        {
            if(check_args(optarg, REG_LINE_COLOR,
REG_LINE_COLOR_EXTENDED))
                sscanf(optarg, "%d.%d.%d", &fill_color[0],
&fill_color[1], &fill_color[2]);
            if(check_args(optarg, REG_LINE_COLOR,
REG_LINE_COLOR_EXTENDED) == 2)
                extend_command_message("--fill_color");
            if(!check_args(optarg, REG_LINE_COLOR,
REG_LINE_COLOR_EXTENDED) || (fill_color[0] > 255 || fill_color[1] >
255 || fill_color[2] > 255))
            {

```

```

        puts(INVALID_COMMAND);
        exit(ERROR_CODE);
    }
    fill_flag = 1;
    cnt_commands+=2;
    break;
}
default:
{
    puts(INVALID_COMMAND);
    exit(ERROR_CODE);
}
}
opt = getopt_long(argc, argv, opts, longopts, &longidx);
}

if(!file_name_flag)
{
    if(check_file_str(argv[argc-1]) && cnt_commands == argc - 2)
        file_name = argv[argc-1];
    else
    {
        puts(FILE_NAME_FORGOTTEN);
        exit(ERROR_CODE);
    }
}
if(!strcmp(file_name, new_file_name))
{
    puts(SAME_INP_OUT_FILES);
    exit(ERROR_CODE);
}
if(!check_file_name(file_name))
{
    puts(FILE_NAME_ERROR);
    exit(ERROR_CODE);
}
if(!check_bmp(file_name))
{
    puts(FILE_TYPE_ERROR);
    exit(ERROR_CODE);
}

array = read_bmp(file_name, &bmfh, &bmif);
if(info_flag)
{
    print_file_header(bmfh);
    printf("\n");
    print_info_header(bmif);
}
if(line_flag)
{
    if(start_flag && end_flag && color_flag && thickness_flag)
    {
        draw_line(array, bmif.height, bmif.width, x0, y0, x1, y1,
color, (thickness-1)/2);
    }
}

```



```

        if(thickness % 2 == 0)
        {
            if(fabs((float)(x0 - x1)) <= fabs((float)(y0 - y1)))
                draw_line(array, bmif.height, bmif.width, x0+1, y0,
x1+1, y1, color, (thickness-1)/2);
            else
                draw_line(array, bmif.height, bmif.width, x0, y0+1,
x1, y1+1, color, (thickness-1)/2);
        }
    }
    else
    {
        puts(LINE_COMMAND_ERROR);
        exit(ERROR_CODE);
    }
}
if(inverse_circle_flag)
{
    if(center_flag && rad_flag)
        invert_circle(array, bmif.height, bmif.width, xc, yc, rad);
    else
    {
        puts(INVERSE_CIRCLE_COMMAND_ERROR);
        exit(ERROR_CODE);
    }
}
if(trim_flag)
{
    if(left_flag && right_flag)
        array = trim(array, &bmif, &bmfh, xl, yl, xr, yr);
    else
    {
        puts(INVERSE_CIRCLE_COMMAND_ERROR);
        exit(ERROR_CODE);
    }
}
if(rombus_flag)
{
    if(size_flag && vert_flag && fill_flag)
    {
        romb(array, bmif.height, bmif.width, xv, yv, rb_size,
fill_color);
    }
    else
    {
        puts("SQUARE_ERROR!");
        exit(ERROR_CODE);
    }
}
if(!info_flag)
    write_bmp(new_file_name, array, bmif.height, bmif.width, bmfh,
bmif);

    exit(0);
}

```