

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

**КУРСОВАЯ РАБОТА
по дисциплине «Программирование»
Тема: Обработка изображений**

Студент гр. 3344

Кузнецов Р.А.

Преподаватель

Глазунов С.А.

Санкт-Петербург

2024

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Кузнецов Р.А.

Группа 3344

Тема работы: Обработка изображений.

Исходные данные:

- Программа **обязательно должна иметь CLI** (опционально дополнительное использование GUI).
- Программа должна реализовывать весь следующий функционал по обработке bmp-файла
- 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).
- Каждую подзадачу следует вынести в отдельную функцию, функции сгруппировать в несколько файлов
- Сборка должна осуществляться при помощи make и Makefile или другой системы сборки

Содержание пояснительной записки:

- Содержание
- Введение
- Описание задания
- Описание реализованных функций, структур
- Описание файловой структуры программы
- Описание модульной структуры, сборки программы
- Примеры работы программы
- Примеры ошибок
- Заключение
- Список использованных источников
- Приложение А. Исходный код программы

Предполагаемый объем пояснительной записки:

Не менее 45 страниц.

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 22.05.2024

Дата защиты реферата: 22.05.2024

Студент		Кузнецов Р.А.
Преподаватель		Глазунов С.А.

АННОТАЦИЯ

Данная курсовая работа посвящена разработке программы на языке C для обработки изображений формата BMP. Программа предоставляет возможность выполнения различных операций над изображениями, таких как создание коллажа, отражение заданных областей, рисование пентаграмм и прямоугольников. Основное внимание уделено проверке корректности входных файлов, поддержке различных версий формата BMP, а также правильному выравниванию данных для записи входного файла. Для управления функциональностью программы используется интерфейс командной строки CLI. Проект структурирован на отдельные функции, сгруппированные в несколько файлов, и поддерживает сборку с использованием Make (CMake).

СОДЕРЖАНИЕ

	Введение	6
1.	Описание задания	7
2.	Описание программы	9
2.1.	Реализованные функции, структуры	
2.2.	Файловая структура программы	
2.3	Модульная структура, сборка	
3.	Примеры работы программы	13
4.	Примеры ошибок	20
	Заключение	21
	Список использованных источников	22
	Приложение А. Исходный код программы	23

ВВЕДЕНИЕ

Цель данной работы заключается в разработке программы на языке C для обработки BMP-изображений с использованием интерфейса командной строки CLI.

Чтобы выполнить поставленные цели, необходимо решить следующие задачи:

1. Изучение изображения с форматом BMP.
2. Реализация функций для обработки изображения.
3. Изучение метода реализации интерфейса командной строки CLI.
4. Оптимизация работы функций программы.
5. Обработка возможных ошибок, причин и способов решения таковых.

Возможные методы решения поставленных задач:

1. Разработка функций для чтения и записи BMP-файлов.
2. Создание структур заголовков для данных изображения.
3. Использование getopt для обработки аргументов командной строки.
4. Организация сборки проекта с использованием Makefile.
5. Разделение задач на отдельные функции для каждой операции.

1. ОПИСАНИЕ ЗАДАНИЯ

Программа должна иметь следующие функции по обработке изображений:

Создать коллаж размера $N \times M$ из одного изображения. Флаг для выполнения данной операции: `--collage`. Коллаж представляет собой это же самое изображение повторяющееся $N \times M$ раз.

Количество изображений по “оси” Y. Флаг `--number_y`. На вход принимает число больше 0

Количество изображений по “оси” X. Флаг `--number_x`. На вход принимает число больше 0

Отражение заданной области. Флаг для выполнения данной операции: `--mirror`. Этот функционал определяется:

выбором оси относительно которой отражать (горизонтальная или вертикальная). Флаг `--axis`, возможные значения `'x'` и `'y'`

Координатами левого верхнего угла области. Флаг `--left_up`, значение задаётся в формате `'left.up'`, где `left` – координата по x, `up` – координата по y

Координатами правого нижнего угла области. Флаг `--right_down`, значение задаётся в формате `'right.down'`, где `right` – координата по x, `down` – координата по y

Рисование пентаграммы в круге. Флаг для выполнения данной операции: `--pentagram`. Пентаграмма 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

толщиной линий и окружности. Флаг `--thickness`. На вход принимает число больше 0

цветом линий и окружности. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где `rrr/ggg/bbb` – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)

Рисование прямоугольника. Флаг для выполнения данной операции: `--rect`. Он определяется:

Координатами левого верхнего угла. Флаг `--left_up`, значение задаётся в формате `left.up`, где `left` – координата по `x`, `up` – координата по `y`

Координатами правого нижнего угла. Флаг `--right_down`, значение задаётся в формате `right.down`, где `right` – координата по `x`, `down` – координата по `y`

Толщиной линий. Флаг `--thickness`. На вход принимает число больше 0

Цветом линий. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где `rrr/ggg/bbb` – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)

Прямоугольник может быть залит или нет. Флаг `--fill`. Работает как бинарное значение: флага нет – `false`, флаг есть – `true`.

цветом которым он залит, если пользователем выбран залитый. Флаг `--fill_color` (работает аналогично флагу `--color`)

2. ОПИСАНИЕ ПРОГРАММЫ

2.1. Реализованные функции, структуры

Во время разработки программы были реализованы такие структуры:

1. `rgb` - используется для хранения цвета пикселя.
2. `BmpFileHeader` - используется для хранения общей информации об изображении.
3. `BmpInfoHeader` - используется для хранения подробной информации об изображении.
4. `BMPfile` - используется для объединения информации о файле и массива цветов пикселей.
5. `coords` - используется для хранения координат какой-либо точки.
6. `Arguments` - используется для хранения флагов и значения входящих флагов.
7. `struct option long_options[]` - структура библиотеки `getopt`.

Во время разработки программы были реализованы такие функции:

1. `void drawPixel(BMPfile* bmp, int x, int y, rgb color)` — функция для рисования одного пикселя.
2. `void drawPixel(BMPfile* bmp, int x, int y, rgb color)` — функция для рисования толщины пикселю.
3. `void drawingLine(BMPfile* bmp, coord start, coord end, int thickness, rgb color)` — функция для рисования линии.
4. `void drawRectangle(BMPfile* bmp, coord left_up, coord right_down, int thickness, rgb color, int fill, rgb fill_color)` — функция для рисования и заливки прямоугольника.
5. `void drawCircle(BMPfile* bmp, int x, int y, int r, int thickness, rgb color)` — функция для рисования окружности.
6. `void drawCircle(BMPfile* bmp, int x, int y, int r, int thickness, rgb color)` — функция для рисования звезду в круге.

7. `BMPfile* loadBMP(char* fname)` — функция записи bmp-файла.
8. `void readRowByRow(FILE* f, BMPfile* bmp)` — функция записи пикселей в массив в правильном порядке.
9. `void writeBMP(char* fname, BMPfile* bmp)` — функция записи bmp-файла.
10. `void freeBMP(BMPfile* bmp)` — функция освобождения памяти.
11. `void check_boundary(int* x_min, int* x_max, int* y_min, int* y_max, int W, int H)` — функция проверки корректности введенных координат.
12. `void mirror(BMPfile* bmp, int axis, coord left_up, coord right_down)` — функция для отражения заданной области.
13. `void collage(BMPfile* bmp, int count_y, int count_x)` — функция создание коллажа NxM из изображения.
14. `int main(int argc, char* argv[])` - главная функция, в которой распределяются основные действия.
15. `void printBMPHeaders(BMPfile* bmp_file)` — функция для печати информации о файле.
16. `void printInfo()` - функция для вывода справки о программе.
17. `void swap(int* a, int* b)` — функция для обмена значения переменных.
18. `int more_zero(int arg)` — функция проверки, что это число 0.
19. `void check_axis(char* optarg)` — проверка на корректность введенной оси.
20. `void another_arguments(char* arg, char* name)` — проверка на то, что флаг не принимает аргументы.
21. `void check_bmp(char* name)` — проверка на bmp-файл.
22. `void check_coord(char* coords)` — проверка на корректность координат.
23. `void more_then_zero(int arg)` — проверка на то, что число больше чем ноль.
24. `void less_then(int arg)` — проверка на то, что число отрицательное.

25. `void check_digit(char * distance)` — проверка на число
26. `int* parse_color(char* color)` — функция создания корректного формата цвета.
27. `void check_color(char *color)` — функция проверки корректности цвета.
28. `Arguments get_arguments(int argc, char* argv[])` - функция реализующая интерфейс командной строки CLI.

2.2. Файловая структура программы

Во время разработки программа была разбита на следующие файлы:

- `Makefile` — файл, который производит компиляцию и сборку проекта.
- `bmp_draw.c` - файл, содержащий функции для рисования на изображении
- `bmp_reader.c` - файл, содержащий функции считывания, записи `bmp`-файла.
- `bmp_reader.h` — заголовочный файл, содержащий прототипы функций, считывания, записи и основные структуры для хранения информации о `bmp`-файле.
- `change_bmp.c` - файл, содержащий функции для какого-либо изменения файла, не включая рисование.
- `main.c` — главный файл, на котором завязаны остальные.
- `print_func.c` - файл, содержащий функции печати справки или информации о файле.
- `processing_bmp.h` - заголовочный файл, содержащий прототипы функций как для изменения файлы, так и для рисования.
- `processing_data.c` - файл, содержащий код для интерфейса командной строки CLI..
- `processing_data.h` - заголовочный файл, содержащий прототипы функций для интерфейса командной строки CLI.

2.3. Модульная структура, сборка

Для сборки проекта используется Makefile:

- переменная CC задает компилятор, который будет использоваться для сборки проекта
- переменная CFLAGS задает флаги компиляции
- переменная TARGET задает имя выходного файла, который будет создан в результате сборки
- переменная SRCS содержит список всех исходных файлов, которые нужно скомпилировать
- переменная OBJS преобразует список исходных файлов в список объектных файлов
- переменная HDRS содержит список заголовочных файлов
- all: \$(TARGET) правило по умолчанию all зависит от целевого файла, выполняется, если просто запустить make без аргументов
- \$(TARGET): \$(OBJS) правило указывает, что целевой файл TARGET зависит от всех объектных файлов OBJS, создает целевой файл
- %.o: %.c \$(HDRS) это правило указывает, как скомпилировать каждый файл .c в соответствующий файл .o
- clean это правило предназначено для очистки проекта.

3. ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

Исходная картинка(bmpfile.bmp):



1. Создание коллажа:

Входные данные:

```
./cw --number_x 2 --collage --input bmpfile.bmp --number_y 2
```

Обработанное изображение:



2. Отражение области:

Входные данные:

```
./cw --axis y --left_up 100.200 --right_down 600.500 --mirror --input bmpfile.bmp
```

Обработанное изображение:



3. Рисование пентаграммы в круге:

Входные данные:

```
./cw --radius 100 --pentagram --input bmpfile.bmp --thickness 5 --center 300.250 -  
-color 255.0.200
```

Обработанное изображение:



4. Рисование прямоугольника:

Входные данные:

```
./cw --rect --thickness 6 --fill --fill_color 200.100.50 --right_down 300.200 --  
left_up 100.50 --color 255.100.0 bmpfile.bmp
```

Обработанное изображение:



5. Вывод справки:

Входные данные:

`./cw --help`

Вывод:

```

                                *Program's options*
=  -i --info <file>             Input file name
=  -o --output <file>          Output file name
=  -I --info                   Show file information
=  -h --help                   Show help about the program
-----
=  --collage                   Create a collage of size N*M from one image
=
=  *-----*Flags for collage*-----*
=  --number_y <value>          Number of images along the Y axis
=  --number_x <value>          Number of images along the X axis
-----
=  --mirror                    Mirroring a specified area
=
=  *-----*Flags for mirror*-----*
=  --axis <value>              Selecting the axis relative to which to reflect
=  --left_up <value>.<value>    x, y left upper corner
=  --right_down <value>.<value> x, y lower right corner
-----
=  --pentagram                 Drawing a pentagram in a circle
=
=  *-----*Flags for pentagram*-----*
=  --center <value>.<value>      x, y center of the pentagram
=  --radius <value>             Radius of the pentagram
=  --thickness <value>          Line's thickness
=  --color <value>.<value>.<value> Line's color
-----
=  --rect                      Drawing a rectangle
=
=  *-----*Flags for rectangle*-----*
=  --left_up <value>.<value>      x, y left upper corner
=  --right_down <value>.<value>   x, y lower right corner
=  --thickness <value>          Line's thickness
=  --color <value>.<value>.<value> Line's color
=  --fill                      Turning shape fill on or off
=  --fill_color <value>.<value>.<value> Shape's fill color
-----
```

6. Вывод информации о изображении:

Входные данные:

```
./cw --info --input bmpfile.bmp
```

Вывод программы:

```
Идентификатор файла(file's signature): BM
Размер файла в байтах(заголовки, данные): 818058
Смещение пиксельных данных в байтах(начальный адрес байта, в котором находится изображение): 138
Размер заголовочной структуры в байтах: 124
Ширина изображения в пикселях: 640
Высота изображения в пикселях: 426
Для каждого пикселя количество цветовых плоскостей: 1
Количество бит на пиксель: 24
Признак наличия сжатия: 0
Размер пиксельных данных в байтах: 817920
Количество пикселей на метр по высоте: 0
Количество пикселей на метр по ширине: 0
Количество цветов в палитре: 0
Количество 'важных' цветов: 0
```

4. ПРИМЕРЫ ОШИБОК

1. Некорректный флаг:

```
romeowku@romeowku:~/Рабочий стол/cw/cw on c$ ./cw --info --str
Course work for option 5.6, created by Roman Kuznetsov.
./cw: unrecognized option '--str'
Invalid argument
```

2. Отсутствие аргумента у флага:

```
romeowku@romeowku:~/Рабочий стол/cw/cw on c$ ./cw --rect --thickness 6 --fill --fill_color 200.100.50 --right_down 300.
200 --left_up 100.50 --color bmpfile.bmp
Course work for option 5.6, created by Roman Kuznetsov.
Incorrect form of color
Use --color <value>.<value>.<value>
```

3. Подача аргумента флагу, который их не принимает:

```
romeowku@romeowku:~/Рабочий стол/cw/cw on c$ ./cw --rect --thickness 6 --fill 1 --fill_color 200.100.50 --right_down 300.
0.200 --left_up 100.50 --color 255.100.0 bmpfile.bmp
Course work for option 5.6, created by Roman Kuznetsov.
--fill don't have arguments
```

4. Одинаковые имена входящего и выходящего изображения:

```
romeowku@romeowku:~/Рабочий стол/cw/cw on c$ ./cw --rect --thickness 6 --fill --fill_color 200.100.50 --right_down 300.
200 --left_up 100.50 --color 255.0.0 --input bmpfile.bmp --output bmpfile.bmp
Course work for option 5.6, created by Roman Kuznetsov.
The input and output file names must not be the same
```

5. Некорректный аргумент для любого флага:

```
romeowku@romeowku:~/Рабочий стол/cw/cw on c$ ./cw --collage --number_y 20.10 --number_x 10/0
Course work for option 5.6, created by Roman Kuznetsov.
Incorrect form of value
```

ЗАКЛЮЧЕНИЕ

В данной курсовой работе была разработана программа на языке C для обработки bmp-изображений с использованием интерфейса командной строки (CLI). Процесс разработки включал в себя реализацию функций для чтения, записи и обработки bmp-файлов, создание структур для представления данных изображения и функций для изменения изображения, а также использование библиотеки getopt для обработки аргументов командной строки. Сборка проекта осуществлялась с помощью Makefile, что обеспечило удобство и автоматизацию процесса компиляции и линковки. В результате выполнения курсовой работы были получены ценные навыки работы с форматом bmp.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Базовые сведения к выполнению курсовой работы по дисциплине «программирование». второй семестр: учеб.-метод. Пособие сост. А. А. Лисс, С. А. Глазунов, М. М. Заславский, К. В. Чайка и др. СПб.: Изд-во СПбГЭТУ "ЛЭТИ", 2024. 36 с.
2. Керниган, Ритчи: Язык программирования С: классическая книга по языку программирования С сост. Керниган Брайан, Ритчи Деннис. США: Издательство Вильямс, 2019 г.
3. The GNU C Library Reference Manual. GETOPT. URL: https://www.gnu.org/software/libc/manual/html_node/Getopt.html (дата обращения 18.05.24)

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

bmp_draw.c

```
#include "processing_bmp.h"

void drawPixel(BMPfile* bmp, int x, int y, rgb color)
{
    if (!(x < 0 || y < 0 || x >= (int)bmp->bmpih.width || y >= (int)bmp->bmpih.height))
        bmp->data[y][x] = color;
}

void drawThickness(BMPfile *bmp, int x, int y, int thickness, rgb color)
{
    int W = bmp->bmpih.width;
    int H = bmp->bmpih.height;
    //итерируемся по диапазону, который охватывает пиксели с учетом
толщины
    for (int j = -thickness / 2; j <= thickness / 2; j++) { //половина толщины в
стороны
        for (int i = -thickness / 2; i <= thickness / 2; i++) {
            //находится ли текущий пиксель в пределах изображения
            if (!(x + i < 0 || y + j < 0 || x + i >= W || y + j >= H)) {
                //если текущий пиксель в пределах толщины добавляем
                if (i * i + j * j <= (thickness / 2) * (thickness / 2)) bmp->data[y + j][x + i] =
color;
            }
        }
    }
}

void drawingLine(BMPfile* bmp, coord start, coord end, int thickness, rgb color)
{
    int x = start.x;
    int y = start.y;

    int x_end = end.x;
    int y_end = end.y;

    int dx = abs(x_end - x); //разница по оси x и y между начальной и конечной
точкой
    int dy = abs(y_end - y);

    int orient_x = x < x_end ? 1 : -1; //направление движения по осям x и y
    int orient_y = y < y_end ? 1 : -1;
```

```

int error = dx - dy; //переменную для отслеживания ошибки(отклонение по
курсу)
int error2;

while (x != x_end || y != y_end) {
    drawPixel(bmp, x, y, color);
    if ((x < (int)bmp->bmpih.width && y < (int)bmp->bmpih.height && x >= 0
&& y >= 0) || (x + (thickness / 2) < (int)bmp->bmpih.width &&
    y + (thickness / 2) < (int)bmp->bmpih.height && x + (thickness / 2) >= 0 &&
y + (thickness / 2) >= 0) || (x - (thickness / 2) < (int)bmp->bmpih.width &&
    y - (thickness / 2) < (int)bmp->bmpih.height && x - (thickness / 2) >= 0 &&
y - (thickness / 2) >= 0)) {
        if (thickness % 2 == 0) drawThickness(bmp, x, y, thickness, color);
        else if (thickness == 1) drawPixel(bmp, x, y, color);
        else drawThickness(bmp, x, y, thickness + 1, color);
    }
    error2 = 2 * error;
    //ошибка и перемещение по оси x
    if (error2 > -dy) {
        error -= dy;
        x += orient_x;
    }
    //ошибка и перемещение по оси y
    if (error2 < dx) {
        error += dx;
        y += orient_y;
    }}

void drawRectangle(BMPfile* bmp, coord left_up, coord right_down, int
thickness, rgb color, int fill, rgb fill_color)
{
    coord left_down = {left_up.x, right_down.y};
    coord right_up = {right_down.x, left_up.y};

    if (fill == 1) {
        for (int y = left_up.y; y < right_down.y + 1; y++) {
            for (int x = left_up.x; x < right_down.x + 1; x++) drawPixel(bmp, x, y,
fill_color);
        }

        drawingLine(bmp, left_up, right_up, thickness, color);
        drawingLine(bmp, left_down, left_up, thickness, color);
        drawingLine(bmp, right_up, right_down, thickness, color);
        drawingLine(bmp, right_down, left_down, thickness, color);
    }
}

```



```
}
```

```
void drawCircle(BMPfile* bmp, int x, int y, int r, int thickness, rgb color)
```

```
{
```

```
    int x0 = r;
```

```
    int y0 = 0;
```

```
    int error = 1 - r;
```

```
    while (x0 >= y0) {
```

```
        // пиксели для текущего радиуса с учетом толщины линии
```

```
        drawThickness(bmp, x + x0, y + y0, thickness, color);
```

```
        drawThickness(bmp, x + y0, y + x0, thickness, color);
```

```
        drawThickness(bmp, x - y0, y + x0, thickness, color);
```

```
        drawThickness(bmp, x - x0, y + y0, thickness, color);
```

```
        drawThickness(bmp, x - x0, y - y0, thickness, color);
```

```
        drawThickness(bmp, x - y0, y - x0, thickness, color);
```

```
        drawThickness(bmp, x + y0, y - x0, thickness, color);
```

```
        drawThickness(bmp, x + x0, y - y0, thickness, color);
```

```
        // координаты и ошибка
```

```
        if (error <= 0) {
```

```
            y0 += 1;
```

```
            error += 2 * y0 + 1;
```

```
        }
```

```
        if (error > 0) {
```

```
            x0 -= 1;
```

```
            error -= 2 * x0 + 1;
```

```
        }
```

```
    }
```

```
}
```

```
void drawPentagram(BMPfile* bmp, int x, int y, int r, int thickness, rgb color)
```

```
{
```

```
    drawCircle(bmp, x, y, r, thickness, color);
```

```
    coord nodes[5];
```

```
    for (int i = 0; i < 5; i++) {
```

```
        double angle = (2 * 3.1415 / 5) * i + -3.1415 / 5;
```

```
        nodes[i].x = round(x + r * sin(angle));
```

```
        nodes[i].y = round(y + r * cos(angle));
```

```
    }
```

```

    for (int i = 0; i < 5; i++) drawingLine(bmp, nodes[i], nodes[(i + 2) % 5],
thickness, color); // здесь nodes[(i+ 1) % 5] для пятиугольника
}

```

bmp_reader.c

```
#include "bmp_reader.h"
```

```
BMPfile* loadBMP(char* fname)
```

```

{
    FILE* f = fopen(fname, "rb");
    if (!f) {
        printf("Ошибка загрузки файла '%s'\n", fname);
        exit(0);
    }

```

```
    BMPfile* bmp = (BMPfile*)malloc(sizeof(BMPfile));
```

```

    /*if (bmp->bmpih.comp != 0 || bmp->bmpih.bits_per_pixel != 24 || (*(unsigned
short*)bmp->bmpfh.ID) != 0x4D42) {
        printf("Предоставленная версия bmp-файла не поддерживается
программой.\nВведите флаг --help чтобы получить информацию о файле.\n");
        exit(0);
    }*/

```

```

    fread(&bmp->bmpfh, sizeof(BmpFileHeader), 1, f);
    fread(&bmp->bmpih, sizeof(BmpInfoHeader), 1, f);

```

```

    bmp->data = (rgb**)malloc(bmp->bmpih.height * sizeof(rgb*)); //кол-во строк
    for(unsigned int i = 0; i < bmp->bmpih.height; i++) {
        bmp->data[i] = (rgb*)malloc(bmp->bmpih.width * sizeof(rgb));
    }

```

```
    readRowByRow(f, bmp);
```

```

    fclose(f);
    return bmp;
}

```

```
//-----
```

```
void readRowByRow(FILE* f, BMPfile* bmp)
```

```

{
    int bytes_per_pixel = bmp->bmpih.bits_per_pixel / 8; // кол-во байт на пиксель
    int row_size = bytes_per_pixel * bmp->bmpih.width; // размер одной строки
    пикселей в байтах

```

```

int row_padding = (4 - (row_size % 4)) % 4; // кол-во байт, необходимых для
выравнивания
unsigned int rows_written = 0; // кол-во записанных в память строк
unsigned char* row = (unsigned char*)malloc(row_size + row_padding); //
память для строк изображения с учетом выравнивания

```

```

fseek(f, bmp->bmpfh.offset_pixels, SEEK_SET); // переходим к началу
изображения

```

```

while(rows_written < bmp->bmpih.height) {
    fread(row, row_size + row_padding, 1, f); // считываем строку с учетом
выравнивания

```

```

    for(int i = 0; i < row_size; i += bytes_per_pixel) { // считываем наоборот
        int pixel_index = i / bytes_per_pixel;
        bmp->data[bmp->bmpih.height - 1 - rows_written][pixel_index].red = row[i +
2];
        bmp->data[bmp->bmpih.height - 1 - rows_written][pixel_index].green = row[i
+ 1];
        bmp->data[bmp->bmpih.height - 1 - rows_written][pixel_index].blue = row[i];

```

```

    }
    rows_written++;
}
free(row);
}

```

```

//-----

```

```

void writeBMP(char* fname, BMPfile* bmp)

```

```

{
    FILE* f = fopen(fname, "wb");
    if (!f) {
        printf("Ошибка загрузки файла \\'%s\\'\n", fname);
        exit(0);
    }

```

```

    fwrite(&bmp->bmpfh, sizeof(BmpFileHeader), 1, f);
    fwrite(&bmp->bmpih, sizeof(BmpInfoHeader), 1, f);

```

```

    int bytes_per_pixel = bmp->bmpih.bits_per_pixel / 8;
    int row_size = bytes_per_pixel * bmp->bmpih.width;
    int row_padding = (4 - (row_size % 4)) % 4;
    unsigned int rows_written = 0;

```

```

    unsigned char* row = (unsigned char*)malloc(row_size + row_padding);

```

```

fseek(f, bmp->bmpfh.offset_pixels, SEEK_SET);

while(rows_written < bmp->bmpih.height) {
    for(int i = 0; i < row_size; i += bytes_per_pixel) {
        int pixel_index = i / bytes_per_pixel;
        row[i + 2] = bmp->data[bmp->bmpih.height - 1 - rows_written][pixel_index].red;
        row[i + 1] = bmp->data[bmp->bmpih.height - 1 - rows_written][pixel_index].green;
        row[i] = bmp->data[bmp->bmpih.height - 1 - rows_written][pixel_index].blue;

    }
    fwrite(row, row_size + row_padding, 1, f);
    rows_written++;
}
free(row);
}
//-----
void freeBMP(BMPfile* bmp)
{
    if (bmp) {
        for(unsigned int i = 0; i < bmp->bmpih.height; i++) free(bmp->data[i]);
        free(bmp->data);
        free(bmp);
    }
}

```

bmp_reader.h

```

#ifndef BMP_READER_H
#define BMP_READER_H
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <math.h>
#include <getopt.h>
#include <regex.h>

#pragma pack(push, 1)
typedef struct BmpFileHeader { // (информация) не зависит от конкретного изображения
    unsigned char ID[2]; //идентификатор файла(file's signature)
    unsigned int file_size; //размер файла в байтах(заголовки, данные)

```

```

    unsigned char unused[4]; // little endian
    unsigned int offset_pixels; //смещение(начальный адрес байта, в котором
находится изображение)
} BmpFileHeader;

```

```

typedef struct BmpInfoHeader { // информация о изображении конкретном
    unsigned int header_size; // чисто изображение, без заголовков ((w * h * bpp)
// 8 (байты))
    unsigned int width; // ширина
    unsigned int height; // высота
    unsigned short color_planes; // для каждого пикселя количество цветовых
плоскостей
    unsigned short bits_per_pixel; // битов на пиксель(качество, количество
цветов)
    unsigned int comp; // признак наличия сжатия(метод сжатия(есть нет))
    unsigned int data_size; // необработанные данные (выравнивание сторк
пикселей)
    unsigned int pwidth; // разрешение для печати ширина
    unsigned int pheight; // разрешение для печати высота
    unsigned int colors_count; // количество цветов в палитре
    unsigned int imp_colors_count; // количество важных цветов (0 все важны)
} BmpInfoHeader;

```

```

typedef struct RGB {
    unsigned char red;
    unsigned char green;
    unsigned char blue;
} rgb;

```

```

typedef struct BMPfile {
    BmpFileHeader bmpfh;
    BmpInfoHeader bmpih;
    rgb** data;
} BMPfile;

```

```

typedef struct coord{
    int x, y;
} coord;

```

```

#pragma pack(pop)

```

```

BMPfile* loadBMP(char* fname);

```

```

void readRowByRow(FILE* f, BMPfile* bmp_file);
void writeBMP(char* fname, BMPfile* bmp_file);
void freeBMP(BMPfile* bmp_file);
#endif

```

change_bmp.c

```
#include "processing_bmp.h"
```

```
void check_boundary(int* x_min, int* x_max, int* y_min, int* y_max, int W, int H)
```

```
{
    if (*x_min < 0 && *y_min < 0) {
        *x_min = 0;
        *y_min = 0;
    } else if (*x_min < 0) {
        *x_min = 0;
    } else if (*y_min < 0) {
        *y_min = 0;
    }
}
```

```
if (*x_max < 0 && *y_max < 0) {
    *x_max = 0;
    *y_max = 0;
} else if (*x_max < 0) {
    *x_max = 0;
} else if (*y_max < 0) {
    *y_max = 0;
}
```

```
if (*x_min > W) {
    *x_min = W - 1;
}
```

```
if (*y_min > H) {
    *y_min = H - 1;
}
```

```
if (*x_max > W) {
    *x_max = W - 1;
}
```

```
if (*y_max > H) {
    *y_max = H - 1;
}
```

```
}
```

```
void mirror(BMPfile* bmp, int axis, coord left_up, coord right_down)
```

```

{
    int x_min = left_up.x;
    int y_min = left_up.y;
    int x_max = right_down.x;
    int y_max = right_down.y;
    int W = bmp->bmpih.width;
    int H = bmp->bmpih.height;

    check_boundary(&x_min, &x_max, &y_min, &y_max, W, H);

    rgb** temp = (rgb**)malloc((H) * sizeof(rgb*));
    if (!temp) {
        printf("Error of allocate height of zone\n");
        exit(0);
    }
    for(int i = 0; i < H; i++) {
        temp[i] = (rgb*)malloc((W) * sizeof(rgb));
        if (!temp[i]) {
            printf("Error of allocate width of zone\n");
            exit(0);
        }
    }

    for (int y = 0; y < H; y++) {
        for (int x = 0; x < W; x++) {
            temp[y][x] = bmp->data[y][x];
        }
    }

    if (axis == 121) { //121 (ось y)
        for (int y = y_min; y <= y_max; y++) {
            for (int x = x_max; x >= x_min; x--) {
                bmp->data[y][x] = temp[y_max - y + y_min][x];
            }
        }
    }
    else if (axis == 120) { //120 (ось x)
        for (int y = y_max; y >= y_min; y--) {
            for (int x = x_min; x <= x_max; x++) {
                bmp->data[y][x] = temp[y][x_max - x + x_min];
            }
        }
    }
}

```

```

for(int i = 0; i < H; i++) {
    free(temp[i]);
}
free(temp);
}

void collage(BMPfile* bmp, int count_y, int count_x)
{

    int width = bmp->bmpih.width;
    int height = bmp->bmpih.height;

    int new_x = bmp->bmpih.width * count_x;
    int new_y = bmp->bmpih.height * count_y;

    // новое изображение
    BMPfile coll_bmp;
    coll_bmp.bmpfh = bmp->bmpfh;
    coll_bmp.bmpih = bmp->bmpih;
    coll_bmp.bmpih.width = new_x;
    coll_bmp.bmpih.height = new_y;
    coll_bmp.bmpfh.file_size += new_x * new_y * sizeof(rgb);
    coll_bmp.data = (rgb**)malloc(new_y * sizeof(rgb*));

    for (int i = 0; i < new_y; i++) {
        coll_bmp.data[i] = (rgb*)malloc(new_x * sizeof(rgb));
        if (coll_bmp.data[i] == NULL) {
            for (int j = 0; j < i; ++j) free(coll_bmp.data[j]);
            free(coll_bmp.data);
            exit(0);
        }
    }

    for (int i = 0; i < new_y; i++) {
        for (int j = 0; j < new_x; j++) {
            int curr_y = i % height;
            int curr_x = j % width;
            coll_bmp.data[i][j] = bmp->data[curr_y][curr_x];
        }
    }

    // замена исходного изображения
    free(bmp->data[0]);
    free(bmp->data);
}

```



```

bmp->bmpih = coll_bmp.bmpih;
bmp->bmpfh = coll_bmp.bmpfh;
bmp->data = coll_bmp.data;

/*for (int i = 0; i < new_y; i++) free(coll_bmp.data[i]);
free(coll_bmp.data);*/
}

```

main.c

```

#include "bmp_reader.h"
#include "print_func.h"
#include "processing_bmp.h"
#include "processing_data.h"

int main(int argc, char* argv[])
{
    printf("Course work for option 5.6, created by Roman Kuznetsov.\n");
    Arguments args = get_arguments(argc, argv);

    if(!args.in_flag) {
        args.in_flag = 1;
        args.in_name = argv[argc - 1];
    }

    if (strcmp(args.in_name, args.out_name) == 0) {
        printf("The input and output file names must not be the same\n");
        exit(41);
    }

    BMPfile* bmp = loadBMP(args.in_name);

    if(args.flag_info == 1) {
        printBMPHeaders(bmp);
        writeBMP(args.out_name, bmp);
        freeBMP(bmp);
        return 0;
    }

    if (args.left_up.x > args.right_down.x) {
        swap(&args.left_up.x, &args.right_down.x);
    }
    if (args.left_up.y > args.right_down.y) {
        swap(&args.left_up.y, &args.right_down.y);
    }
}

```

```

if (args.flag_collage == 1 && args.flag_num_x == 1 && args.flag_num_y == 1)
{
    if (more_zero(args.num_y) == 1 && more_zero(args.num_x) == 1) {
        writeBMP(args.out_name, bmp);
        freeBMP(bmp);
        return 0;
    } else if (more_zero(args.num_y) == 1 || more_zero(args.num_x) == 1) {
        printf("Incorrect value\nNum_y || num_x can't be less than zero\n");
        exit(43);
    } else collage(bmp, args.num_y, args.num_x);

    } else if (args.flag_mirror == 1 && args.flag_axis == 1 && args.flag_left_up ==
1 && args.flag_right_down == 1) {
        mirror(bmp, args.axis, args.left_up, args.right_down);

    } else if (args.flag_center == 1 && args.flag_radius == 1 && args.flag_thickness
== 1 && args.flag_color == 1) {
        drawPentagram(bmp, args.center[0], args.center[1], args.radius, args.thickness,
args.color);

    } else if (args.flag_left_up == 1 && args.flag_right_down == 1 &&
args.flag_thickness == 1 && args.flag_color == 1) {
        drawRectangle(bmp, args.left_up, args.right_down, args.thickness, args.color,
args.fill, args.fill_color);

    } else {
        printf("Incorrect flag\n");
        exit(42);
    }
    writeBMP(args.out_name, bmp);

    freeBMP(bmp);
    return 0;
}

```

Makefile

CC = gcc

CFLAGS = -Wall -Wextra

TARGET = cw

SRCS = main.c bmp_reader.c print_func.c bmp_draw.c change_bmp.c
processing_data.c

OBJS = \$(SRCS:.c=.o)

HDRS = bmp_reader.h print_func.h processing_bmp.h processing_data.h

all: \$(TARGET)

\$(TARGET): \$(OBJS)

\$(CC) \$(CFLAGS) \$(OBJS) -o \$(TARGET) -lm

%.o: %.c \$(HDRS)

\$(CC) \$(CFLAGS) -c \$< -o \$@

clean:

\$(RM) \$(TARGET) \$(OBJS)

print_func.c

#include "print_func.h"

void printBMPHeaders(BMPfile* bmp_file)

```
{
    //printf("Описание BMP-файла \'%s\':\n", filename);
    printf("-----\n");
    printf("\tИдентификатор файла(file's signature):  %c%c\n", bmp_file-
>bmpfh.ID[0], bmp_file->bmpfh.ID[1]);
    printf("\tРазмер файла в байтах(заголовки, данные):  %u\n", bmp_file-
>bmpfh.file_size);
    printf("\tСмещение пиксельных данных в байтах(начальный адрес байта, в
котором находится изображение): %d\n", bmp_file->bmpfh.offset_pixels);

    printf("\tРазмер заголовочной структуры в байтах: %d\n", bmp_file-
>bmpih.header_size);
    printf("\tШирина изображения в пикселях: %d\n", bmp_file->bmpih.width);
    printf("\tВысота изображения в пикселях: %d\n", bmp_file->bmpih.height);
    printf("\tДля каждого пикселя количество цветовых плоскостей: %d\n",
bmp_file->bmpih.color_planes);
    printf("\tКоличество бит на пиксель: %d\n", bmp_file->bmpih.bits_per_pixel);
    printf("\tПризнак наличия сжатия: %d\n", bmp_file->bmpih.comp);
    printf("\tРазмер пиксельных данных в байтах: %d\n", bmp_file-
>bmpih.data_size);
    printf("\tКоличество пикселей на метр по высоте: %d\n", bmp_file-
>bmpih.pwidth);
    printf("\tКоличество пикселей на метр по ширине: %d\n", bmp_file-
>bmpih.pheight);
    printf("\tКоличество цветов в палитре: %d\n", bmp_file->bmpih.colors_count);
```

```

    printf("\tКоличество      'важных'      цветов:      %d\n",      bmp_file-
>bmpih.imp_colors_count);
    printf("-----
-----\n");
}

void printInfo()
{
    printf("-----
\n");
    printf("      *Program's options*\n");
    printf("=  -i --info <file>      Input file name\n");
    printf("=  -o --output <file>      Output file name\n");
    printf("=  -I --info      Show file information\n");
    printf("=  -h --help      Show help about the program\n");
    printf("-----
\n");
    printf("=  --collage      Create a collage of size N*M from one image\n\n");
    printf("=      *-----*Flags for collage*-----*\n");
    printf("=  --number_y <value>      Number of images along the Y axis\n");
    printf("=  --number_x <value>      Number of images along the X axis\n");
    printf("-----
\n");
    printf("=  --mirror      Mirroring a specified area\n\n");
    printf("=      *-----*Flags for mirror*-----*\n");
    printf("=  --axis <value>      Selecting the axis relative to which to
reflect\n");
    printf("=  --left_up <value>.<value>      x, y left upper corner\n");
    printf("=  --right_down <value>.<value>      x, y lower right corner\n");
    printf("-----
\n");
    printf("=  --pentagram      Drawing a pentagram in a circle\n\n");
    printf("=      *-----*Flags for pentagram*-----*\n");
    printf("=  --center <value>.<value>      x, y center of the pentagram\n");
    printf("=  --radius <value>      Radius of the pentagram\n");
    printf("=  --thickness <value>      Line's thickness\n");
    printf("=  --color <value>.<value>.<value>      Line's color\n");
    printf("-----
\n");
    printf("=  --rect      Drawing a rectangle\n\n");
    printf("=      *-----*Flags for rectangle*-----*\n");
    printf("=  --left_up <value>.<value>      x, y left upper corner\n");
    printf("=  --right_down <value>.<value>      x, y lower right corner\n");
    printf("=  --thickness <value>      Line's thickness\n");

```

```

printf("=  --color <value>.<value>.<value>          Line's color\n");
printf("=  --fill                               Turning shape fill on or off\n");
printf("=  --fill_color <value>.<value>.<value>      Shape's fill color\n");
printf("-----\n");
}

void printBMPPixels(BMPfile* bmp_file)
{
    for(unsigned int y = 0; y < bmp_file->bmpih.height; y++) {
        for(unsigned int x = 0; x < bmp_file->bmpih.width; x++) {
            printf("%.3d  %.3d  %.3d\t ", bmp_file->data[y][x].red, bmp_file->data[y][x].green, bmp_file->data[y][x].blue);
        }
        printf("\n");
    }
}

```

print_func.h

```

#include <stdio.h>
#include <stdlib.h>
#include "bmp_reader.h"
#ifndef PRINT_FUNC_H
#define PRINT_FUNC_H

void printBMPHeaders(BMPfile* bmpf);
void printBMPPixels(BMPfile* bmp_file);
void printInfo();

#endif

```

processing_bmp.h

```

#ifndef PROCESSING_BMP_H
#define PROCESSING_BMP_H
#include "bmp_reader.h"
#include "processing_data.h"

void check_boundary(int* x_min, int* x_max, int* y_min, int* y_max, int W, int H);
void mirror(BMPfile* bmp, int axis, coord left_up, coord right_down);
void collage(BMPfile* bmp, int count_y, int count_x);
void drawPixel(BMPfile* bmp, int x, int y, rgb color);
void drawThickness(BMPfile *bmp, int x, int y, int thickness, rgb color);
void drawingLine(BMPfile* bmp, coord start, coord end, int thickness, rgb color);

```

```

void drawRectangle(BMPfile* bmp, coord left_up, coord right_down, int
thickness, rgb color, int fill, rgb fill_color);
void drawCircle(BMPfile* bmp, int x, int y, int r, int thickness, rgb color);
void drawPentagram(BMPfile* bmp, int x, int y, int r, int thickness, rgb color);

```

```

#endif

```

processing_data.c

```

#include "processing_data.h"

```

```

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

```

```

int more_zero(int arg)
{
    if (arg == 0) return 1;
    return 0;
}

```

```

void check_axis(char* optarg)
{
    if(strcmp(optarg, "x") != 0 && strcmp(optarg, "y") != 0) {
        printf("Invalid syntax axis, can only be x or y\n");
        exit(41);
    }
}

```

```

void another_arguments(char* arg, char* name)
{
    if(arg != NULL){
        if(!strstr(arg, "--")){
            fprintf(stderr, "%s don't have arguments\n", name);
            exit(44);
        }
    }
}

```

```

void check_bmp(char* name)
{
    regex_t regex;

```

```

int reti = regcomp(&regex, "^\\S+\\.?[A-Za-z]*$", REG_EXTENDED);
if (reti) {
    fprintf(stderr, "Could not compile regex\n");
    exit(1);
};
reti = regexec(&regex, name, 0, NULL, 0);
if (reti) {
    fprintf(stderr, "Error, it's not bmp file!\n");
    exit(41);
}
}

void check_coord(char* coords)
{
    regex_t regex;
    int reti = regcomp(&regex, "^\\-?[0-9]+\\.\\.\\-?[0-9]+$", REG_EXTENDED);
    if (reti) {
        fprintf(stderr, "Could not compile regex\n");
        exit(1);
    }
    reti = regexec(&regex, coords, 0, NULL, 0);
    if (reti) {
        fprintf(stderr, "Incorrect coordinates\nUse <value>.<value>\n");
        exit(41);
    }
}

void more_than_zero(int arg)
{
    if (arg <= 0) {
        printf("Incorrect value\nThickness || radius can be more then zero\n");
        exit(41);
    }
}

void less_than(int arg)
{
    if (arg <= -1) {
        printf("Incorrect value\nNum_y || num_x can be less then zero\n");
        exit(41);
    }
}

```

```

void check_digit(char * distance)
{
    regex_t regex;
    int reti = regcomp(&regex, "^\\-?[0-9]+$", REG_EXTENDED);
    if (reti) {
        fprintf(stderr, "Could not compile regex\n");
        exit(1);
    };
    reti = regexec(&regex, distance, 0, NULL, 0);
    if (reti) {
        fprintf(stderr, "Incorrect form of value\n");
        exit(41);
    }
}

int* parse_color(char* color)
{
    int* result = malloc(3 * sizeof(int));
    char* copy_color = strdup(color);
    char* token = strtok(copy_color, ".");
    for(int i = 0; i < 3; i++){
        result[i] = atoi(token);
        token = strtok(NULL, ".");
    }
    free(copy_color);
    return result;
}

void check_color(char *color)
{
    regex_t regex;
    int reti = regcomp(&regex, "^[0-9]+\\. [0-9]+\\. [0-9]+$", REG_EXTENDED);
    if (reti) {
        fprintf(stderr, "Regex's error for color\n");
        exit(1);
    }

    reti = regexec(&regex, color, 0, NULL, 0);
    if (reti){
        fprintf(stderr, "Incorrect form of color\nUse --color
<value>.<value>.<value>\n");
        exit(41);
    }
}

```



```

    int *color_rgb = parse_color(color);
    if (color_rgb[0] > 255 || color_rgb[1] > 255 || color_rgb[2] > 255 || color_rgb[0]
< 0 || color_rgb[1] < 0 || color_rgb[2] < 0) {
        free(color_rgb);
        fprintf(stderr, "Incorrect form of color\nUse --color <0-255>.<0-255>.<0-
255>\n");
        exit(41);
    }
    free(color_rgb);
}

```

```

Arguments get_arguments(int argc, char* argv[])
{
    Arguments args = {.out_name = "out.bmp", .in_name = NULL, .fill = 0};

```

```

    int opt;
    int long_index;
    static struct option long_options[] = {
        {"input", required_argument, NULL, 'i'},
        {"output", required_argument, NULL, 'o'},
        {"info", no_argument, NULL, 'I'},
        {"help", no_argument, NULL, 'h'},
        {"collage", no_argument, NULL, 'c'},
        {"number_y", required_argument, NULL, 'q'},
        {"number_x", required_argument, NULL, 'w'},
        {"mirror", no_argument, NULL, 'm'},
        {"axis", required_argument, NULL, 'e'},
        {"left_up", required_argument, NULL, 't'},
        {"right_down", required_argument, NULL, 'y'},
        {"pentagram", no_argument, NULL, 'p'},
        {"center", required_argument, NULL, 'a'},
        {"radius", required_argument, NULL, 's'},
        {"thickness", required_argument, NULL, 'u'},
        {"color", required_argument, NULL, 'd'},
        {"rect", no_argument, NULL, 'r'},
        {"fill", no_argument, NULL, 'f'},
        {"fill_color", required_argument, NULL, 'g'},
        {NULL, 0, NULL, 0}};

```

```

    opt = getopt_long(argc, argv, "i:o:Ihcmprq:w:e:y:a:s:u:d:f:g:", long_options,
&long_index);

```

```

while (opt != -1) {
    switch (opt) {

```

```

case 'i':
    check_bmp(optarg);
    args.in_name = optarg;
    args.in_flag = 1;
    break;
case 'o':
    args.out_name = optarg;
    args.out_flag = 1;
    break;
case 'T':
    args.flag_info = 1;
    break;
case 'h':
    another_arguments(argv[optind], "--help");
    args.flag_help = 1;
    printInfo();
    exit(0);
case 'c':
    another_arguments(argv[optind], "--collage");
    args.flag_collage = 1;
    break;
case 'm':
    another_arguments(argv[optind], "--mirror");
    args.flag_mirror = 1;
    break;
case 'p':
    another_arguments(argv[optind], "--pentagram");
    args.flag_pentagram = 1;
    break;
case 'r':
    another_arguments(argv[optind], "--rect");
    args.flag_rect = 1;
    break;
case 'q':
    check_digit(optarg);
    less_than(atoi(optarg));
    args.flag_num_y = 1;
    args.num_y = atoi(optarg);
    break;
case 'w':
    check_digit(optarg);
    less_than(atoi(optarg));
    args.flag_num_x = 1;
    args.num_x = atoi(optarg);

```

```

    break;
case 'e':
    check_axis(optarg);
    args.flag_axis = 1;
    if (strcmp(optarg, "x") == 0) {
        args.axis = 120;
    } else args.axis = 121;
    break;
case 't':
    check_coord(optarg);
    args.flag_left_up = 1;
    sscanf(optarg, "%d.%d", &args.left_up.x, &args.left_up.y);
    break;
case 'y':
    check_coord(optarg);
    args.flag_right_down = 1;
    sscanf(optarg, "%d.%d", &args.right_down.x, &args.right_down.y);
    break;
case 'a':
    check_coord(optarg);
    args.flag_center = 1;
    sscanf(optarg, "%d.%d", &args.center[0], &args.center[1]);
    break;
case 's':
    check_digit(optarg);
    more_than_zero(atoi(optarg));
    args.flag_radius = 1;
    args.radius = atoi(optarg);
    break;
case 'u':
    check_digit(optarg);
    more_than_zero(atoi(optarg));
    args.flag_thickness = 1;
    args.thickness = atoi(optarg);
    break;
case 'd':
    check_color(optarg);
    args.flag_color = 1;
    sscanf(optarg, "%hhu.%hhu.%hhu", &args.color.red, &args.color.green,
&args.color.blue);
    break;
case 'f':
    another_arguments(argv[optind], "--fill");
    args.fill = 1;

```

```

        break;
    case 'g':
        check_color(optarg);
        args.flag_fill_color = 1;
        sscanf(optarg, "%hhu.%hhu.%hhu", &args.fill_color.red,
&args.fill_color.green, &args.fill_color.blue);
        break;
    default:
        printf("Invalid argument\n");
        exit(41);
    }
    opt = getopt_long(argc, argv, "i:o:lhcmprq:w:e:y:u:a:s:d:f:g:", long_options,
&long_index);
    }
    return args;
}

```

processing_data.h

```

#ifndef PROCESSING_DATA_H
#define PROCESSING_DATA_H
#include "bmp_reader.h"
#include "print_func.h"

```

```

typedef struct Arguments {

```

```

    char* out_name;
    char* in_name;
    int out_flag;
    int in_flag;
    int flag_info;
    int flag_help;

```

```

    int flag_num_y;
    int num_y;
    int flag_num_x;
    int num_x;

```

```

    int flag_collage;
    int flag_pentagram;
    int flag_rect;
    int flag_mirror;

```

```

    int flag_axis;
    int axis;
    int flag_left_up;

```

```

coord left_up;
int flag_right_down;
coord right_down;
int flag_center;
int center[2];
int flag_radius;
int radius;
int flag_thickness;
int thickness;
int flag_color;
rgb color;
int flag_fill_color;
rgb fill_color;
int fill;

} Arguments;

void swap(int* a, int* b);
int more_zero(int arg);
void less_then(int arg);
void more_then_zero(int arg);
void check_axis(char* optarg);
void another_arguments(char* arg, char* name);
void check_bmp(char* name);
void check_coord(char* coords);
void check_digit(char * distance);
int* parse_color(char* color);
int* parse_color(char* color);
Arguments get_arguments(int argc, char *argv[]);

#endif

```