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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка BMP файлов

Студентка гр. 2384 _____ Валеева А. А.

Преподаватель _____ Гаврилов А. В.

Санкт-Петербург

2023

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студентка: Валеева А. А.

Группа 2384

Тема работы: Обработка BMP файлов

Исходные данные:

Вариант 1

Программа **должна** иметь CLI или GUI.

Общие сведения

- 24 бита на цвет
- без сжатия
- файл всегда соответствует формату BMP (но стоит помнить, что версий у формата несколько)
- обратите внимание на выравнивание; мусорные данные, если их необходимо дописать в файл для выравнивания, должны быть нулями.
- обратите внимание на порядок записи пикселей
- все поля стандартных BMP заголовков в выходном файле должны иметь те же значения что и во входном (разумеется кроме тех, которые должны быть изменены).

Программа должна реализовывать весь следующий функционал по обработке bmp-файла

1. Рисование прямоугольника. Он определяется:
    - Координатами левого верхнего угла
    - Координатами правого нижнего угла
    - Толщиной линий
    - Цветом линий
    - Прямоугольник может быть залит или нет

- цветом которым он залит, если пользователем выбран залитый

2. Сделать рамку в виде узора. Рамка определяется:
    - Узором (должно быть несколько на выбор. Красивый узор можно получить используя фракталы)
    - Цветом
    - Шириной
3. Поворот изображения (части) на 90/180/270 градусов. Функционал определяется
    - Координатами левого верхнего угла области
    - Координатами правого нижнего угла области
    - Углом поворота
4. Рисование окружности. Окружность определяется:
    - **либо** координатами левого верхнего и правого нижнего угла квадрата, в который она вписана, **либо** координатами ее центра и радиусом
    - толщиной линии окружности
    - цветом линии окружности
    - окружность может быть залитой или нет
    - цветом которым залита сама окружность, если пользователем выбрана залитая окружность

Содержание пояснительной записки:
«Аннотация», «Содержание», «Введение», «Описание выполнения работы», «Программная реализация функционала», «Заключение», «Список использованных источников», «Примеры работы кода», «Исходный код».

Предполагаемый объем пояснительной записки:
Не менее 15 страниц.

Дата выдачи задания: 22.03.2023

Дата сдачи реферата: 25.05.2023

Дата защиты реферата: 27.05.2023

Студентка _____ Валеева А. А.

Преподаватель _____ Гаврилов А. В.

АННОТАЦИЯ

КВ ходе выполнения курсовой работы была создана программа на языке программирования C, которая обрабатывает BMP-файл. Программа имеет CLI(Command Line Interface) с возможностью вывод справки о программе, реализуемых функциях, ключах и их аргументов. Программа поддерживает BMP-файлы 3 версии, глубиной кодирования 24 бита, без сжатия. Разработка велась на операционной системе Linux Ubuntu 22.04 в IDE Clion с использованием компилятора gcc. При вводе некорректных ключей пользователю выводятся ошибки.

Пример работы программы приведен в приложении А.

Исходный код программы приведен в приложении В.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы:

Создать программу на C, которая обрабатывает BMP-файл согласно запросу пользователя в соответствии с заявленным функционалом.

Задачи:

Для выполнения работы необходимо:

1. Создать структуры для работы BMP-файлов
2. Реализовать считывание и запись BMP-файлов
3. Написать функции обработки изображения
4. Написать функции помощи и вывода информации о файле
5. Реализовать обработку запросов пользователя с помощью CLI

Для чтения и записи файлов будут использоваться функции библиотеки stdio.h.

Для реализации консольного интерфейса будет использована библиотека getopt.h.

Для обработки изображения будет использоваться двумерный массив структур-пикселей.

2. ХОД ВЫПОЛНЕНИЯ РАБОТЫ

2.1. Создание и объявление структур.

Так как существует выравнивание структур, в памяти могут образовываться незаполненные ячейки, который могут создать неудобства при считывании заголовка BMP-файла. Чтобы этого избежать, все структуры оборачиваем в директивы *#pragma pack(push, 1)* — размер выравнивания устанавливается в 1 байт, и возвращаемся к исходной настройке через *#pragma pack(pop)*.

Далее создаются структуры *BMPHeader* и *DIPHeader* с полями, соответствующим 3 версии формата BMP. Структура *Arguments* хранит в себе аргументы, используемые в функциях. Поля структуры *RGB* соответствуют каналам RGB-компонент, каждое поле принимает значение от 0 до 255. А массив из таких структур будет хранить картинку.

Создаем структуру *BMP_file* с полями из структур заголовков и двумерным массивом структур *RGB*.

Структуры объявлены в заголовочном файле *bmp_processing.h.*

2.2. Считывание и запись файла.

Для считывания изображения используется функция *read_bmp(),* с помощью функции *fopen* открывается файл с переданным в качестве аргумента именем, если файл не найдет, то выдается ошибка *err_find_file()* и работа программы завершается. Далее выделяем память для структуры файла, выделяется верно — так как выравнивание у структур с помощью директивы установлено размером 1 байт. Через функцию *fread()* заполняем заголовки. Проверим значение переменной *bits_per_pixel*, если оно не 24, то возвращаем ошибку *count_bits()*. Выделим память для хранения двумерного массива пикселей *pixels_arr,* вызовем функцию *fill_img()* (описана далее), закроем файл и вернем переменную типа *BMP_file*.

Следующая функция — *fill_img(),* используется для заполнения массива *pixels_arr.* Заводим переменную *row_size*, равную размеру строки в байтах (в которой хранятся пиксели изображения), а также посчитаем количество байт, которые нужно добавить для выравнивания. С помощью *fseek()* смещаем указатель по смещению на начало массива. Для строки выделяем память с учетом выравнивания, далее циклами заполняем поля массива структур *pixels_arr,* начиная снизу, так как строчки в BMP-файле хранятся в обратном порядке, то есть в начале идет самая нижняя строка изображения. Так же нужно помнить о том, что цвета пикселя хранятся в обратном порядке - *blue, green, red.* Освобождаем выделенную для строки память.

Запись файла вывода производится в функции *output_bmp_file().* Заводим переменную *data,* вызываем функцию *rgb_to_str,* которая преобразует двумерный массив обратно в строку (в *data*). Данные из этой строки мы с помощью функции *fwrite()* записываем в поток *fp.* Освобождаем выделенную память, закрываем файл.

Функции объявлены в заголовочном файле *bmp_processing.h.*

2.3. Функции заданий курсовой работы.

1. Первая функция — *rectangle(),* рисующая прямоугольник. В качестве аргументов получает следующие параметры: координаты левого верхнего угла, правого нижнего, толщину и цвет линии, наличие заливки и ее цвет. Проверяю координаты и цвет на корректность, вывожу ошибки при неверно введенных значениях или при выходе за границу изображения. С помощью трех циклов *for* прохожусь по ширине линии (фиксированный цикл) и по длине и по ширине. Заполняем границы прямоугольника функцией *draw_point(),* заполняющий структуру *RGB* переданными значениями цвета(*red, green, blue*). Если переменная *shape_fill,* отвечающая за наличие заполнения прямоугольника, равна 1, то проходимся по прямоугольнику и вызывая функцию *draw_point(),* закрашиваем точки внутри фигуры.

2. Следующая функция — *turn(),* поворачивающая изображение на 90, 180 и 270 градусов. Передаваемые аргументы: координаты левого верхнего и правого нижнего углов, а так же угол поворота. Проверяем данные на корректность. Поворот на 180 делаем отражением выбранной области сначала по горизонтальной оси, потом по вертикальной.

В повороте на 90 и на 270 возникает проблема, что при повороте прямоугольника нужно проверять, влазит ли он по краям. Поворот производится с помощью создания нового массива *tmp_arr* типа *RGB* в который мы записываем нужный кусочек (или все изображение) и затем в наш изначальный рисунок с помощью правильно высчитанных координат записываем пиксели из массива. Освобождаем выделенную под массив память.

3. Третья функция — *draw_frame(),* рисование рамки. Во флагах передаем число от 1 до 3, отвечающее за узор рамки. Число обрабатывается оператором *switch()* и вызывает одну из трех функций, рисующих разные узоры. Так же передается ширина рамки. Мы создаем новое изображение, закрашенное черным цветом, его ширина и высоты высчитываются с учетом ширины рамки. После рисования рамки мы на новом изображении отрисовываем старое с помощью функции *drow_point().* Освобождаем память от старого изображения и возвращаем новое.

4. Четвертая функция — *circle()*, рисование окружности. Нам требуется реализовать рисование с помощью двух разных переданных флагов. Первый способ: передается радиус и координаты центра. Второй способ: координаты левого нижнего и правого углов. Также нам нужна толщина линии, ее цвет, наличие заливки и ее цвет. При втором способе передачи высчитываем радиус с координаты центра. Проверяем введенные значение на правильность, выводим ошибки, если требуется. Проходимся двумя циклами по длине и ширине картинки, проверяем, находится ли точка внутри окружности следующим образом: если гипотенуза, полученная суммой квадратов расстояния от координат центра до текущей точки, меньше, чем длина радиуса с учетом толщины линии, и при этом переменная *shape_fill* = 1, то тогда закрашиваем

точку с помощью *draw_point()*. Далее закрашиваем контур, если точка находится между длиной радиуса и длинной радиуса + ширина линии.

Функции объявлены в заголовочном файле *cw_functions.h.*

2.4. Вывод справки и информации о файле.

Для вывода справки — помощь была написана функция *help*, которая выводит в терминал описание программы, описание функций и ключи, необходимые для каждой функции, информацию о ключах.

2.5. Считывание пользовательских данных.

Для считывания данных использовалась библиотека getopt. Были созданы короткие и длинные ключи, в которые пользователь передает данные, далее, в зависимости от вызванной функции, программа выполнит задачу и сохранит файл, либо выведет сообщение-ошибку в терминал.

2.6. Создание Makefile.

Программы была разделена на следующие файлы:

bmp_processing.h. – заголовочный файл, в котором содержится объявление структур и функции работы с BMP-файлом.

bmp_processing.c – файл, в котором содержатся функции работы с BMP-файлом.

print.*c* — файл, в котором содержаться вызовы ошибок, а также руководство по использованию программы и информация о BMP-файле.

print.*h* — заголовочный файл, в котором содержаться сигнатуры функций вызова ошибок, а также руководство по использованию программы и информация о BMP-файле.

cli.*c* — *файл*, в котором содержаться функции обработки интерфейсной строки.

cli.*h* — файл, содержащий вызовы функций обработки интерфейсной строки.

cw_functions.c — файл, содержащий функции, используемые для решений заданий курсовой работы;

cw_functions.h — файл, содержащий вызовы функций, используемых для выполнения заданий;

main.c — файл, в котором содержатся функции-исполнители.

Сборка программы осуществляется с помощью Makefile, в котором прописаны все необходимые цели и зависимости, и утилиты make.

ЗАКЛЮЧЕНИЕ

В реультате выполнения курсовой работы была написана программа на языке Си, имеющая CLI, для обработки изображений — BMP-файлов.

Пользователь вводит ключ функции, название файла, который необходимо изменить, ключи, необходимые для выполнения функции. Преобразования, которые может сделать программа: рисование прямоугольника, окружности, рамки и поворот изображения на 90, 180 и 270 градусов.

Полученный результат соответствует поставленной цели.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Информация о BMP-файле и его версиях.
https://ru.wikipedia.org/wiki/BMP
2. Работа с библиотекой getopt.h для работа с CLI/
https://man7.org/linux/man-pages/man3/getopt.3.html

ПРИЛОЖЕНИЕ А
ПРИМЕР РАБОТЫ ПРОГРАММЫ

1. Пример вывода информации о bmp-файле:

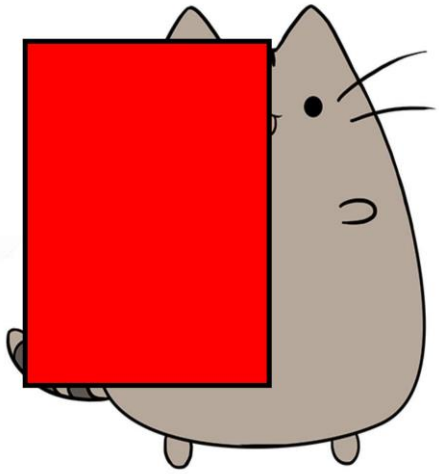
```
alina@Linchik2:~/BMP_Reader$ ./main ./kitten.bmp -i
Информация о bmp-файле:
ID =                    BM
file_size =             3000054
pixel_offset =          54
header_size =           40
width =                 1000
height =                1000
planes =                1
bits_per_pixel =        24
compression =           0
data_size =             3000000
x_pixels_per_meter =            11811
y_pixels_per_meter =            11811
colors_in_color_table =                 0
important_color_count =         0
```

2. Пример вывода справки:

```
alina@Linchik2:~/BMP_Reader$ ./main ./cat.bmp -h
Руководство по использованию программы:
Программа обрабатывает BMP-файлы версии [...]. С глубиной изображения 24 бита на пиксель
Формат ввода: <имя исполняемого файла> <имя BMP-файла для обработки> <функция> -<ключ>/--<полный ключ> <арг
умент> ...
Функции/ключи:
Первый ключ: -g/--g_rectangle-- Рисование прямоугольника.
        -f/--first  [<x-координата>.<y-координата>] - координата левого верхнего угла фигуры.
        -s/--second [<x-координата>.<y-координата>] - координата правого нижнего угла фигуры.
        -w/--width  [<число>] - толщина линии
        -F/--line_color  [<число>.<число>.<число>] - цвет линии (RGB).
        -H/--shape  [<число>] - выбор наличия заливки фигуры (0 - не залит, 1 - залит).
        -S/--shape_color [<число>.<число>.<число>] - цвет заливки (RGB).
Второй ключ: -d/--draw_frame -- Рисование рамки в виде узора.
        -p/--pattern [<число>] - номер узора рамки, число от 1 до 3.
        -F/--line_color  [<число>.<число>.<число>] - цвет рамки(RGB).
        -w/--width  [<число>] - ширина рамки.
Третий ключ: -t/--turn -- Поворот изображения (части) на 90/180/270 градусов.
        -f/--first  [<x-координата>.<y-координата>] - координата левого верхнего угла фигуры.
        -s/--second [<x-координата>.<y-координата>] - координата правого нижнего угла фигуры.
        -a/--angle  [<число>] - угол поворота (90/180/270 градусов).
Четвертый ключ: -c/--circle --Рисование окружности.
        -1/--path_1  [<радиус>.<x-координата>.<y-координата>] - радиус и координата центра окружности.
        -2/--path_2  [<x-координата>.<y-координата>.<x-координата>.<y-координата>] - координаты левого верх
него и правого нижнего углов фигуры.
        -w/--width  [<число>] - толщина линии
        -F/--line_color  [<число>.<число>.<число>] - цвет линии (RGB).
        -H/--shape  [<число>] - выбор наличия заливки фигуры (0 - не залит, 1 - залит).
        -S/--shape_color [<число>.<число>.<число>] - цвет заливки (RGB).
-h/--help - вывод справки о работе программы
-i/--info - вывод подробной информации о bmp-файле
-o/--output [путь] - файл для вывода (по умолчанию исходный файл, путь - относительный)
```

3. Пример выполнения индивидуального задания №1 — Рисование прямоугольника:

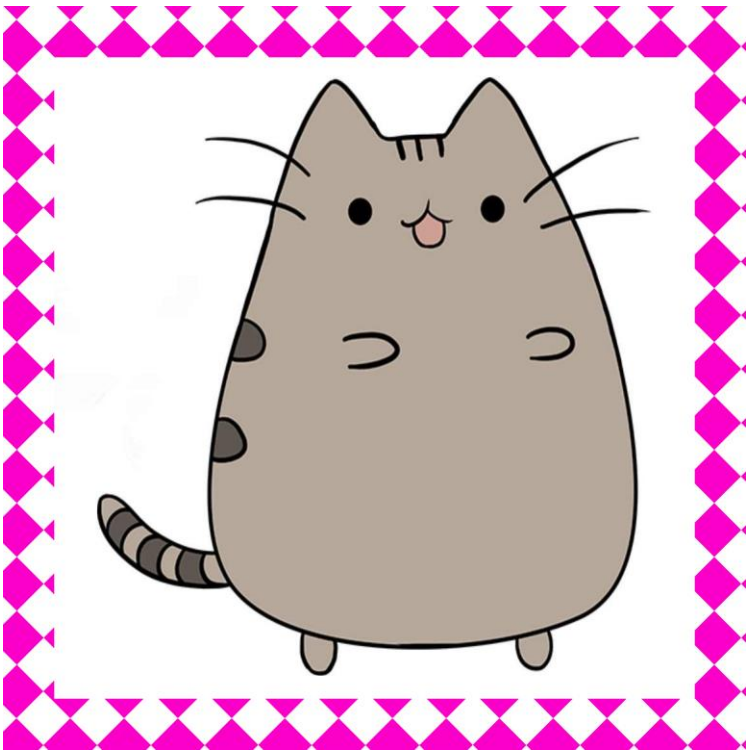
```
./main ./kitten.bmp -g -f 100.100 -s 600.800 -w 10 -F 0.0.0 -H 1 -S 255.0.0
```



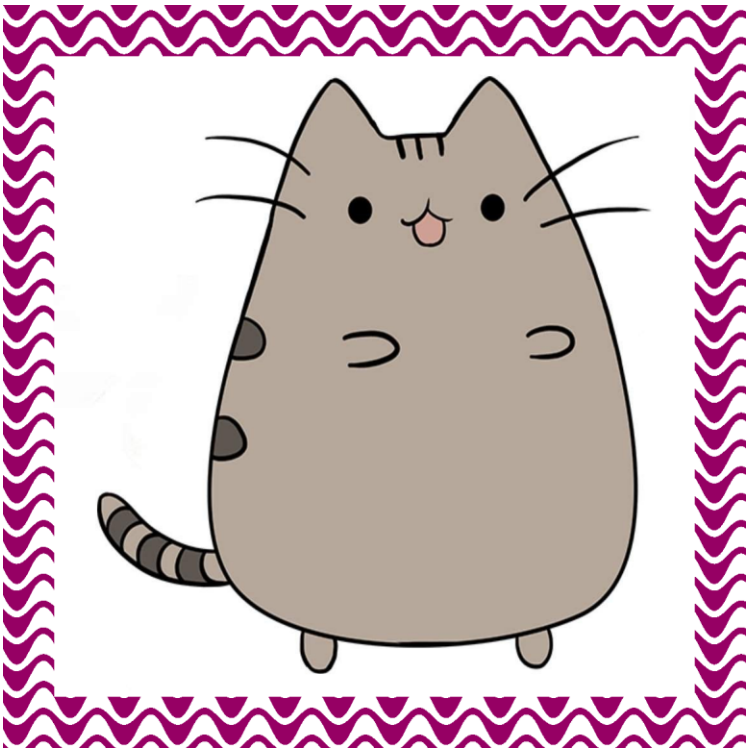
4. Пример выполнения индивидуального задания №2 — Рисование рамки:

Первый узор (pattern = 1):

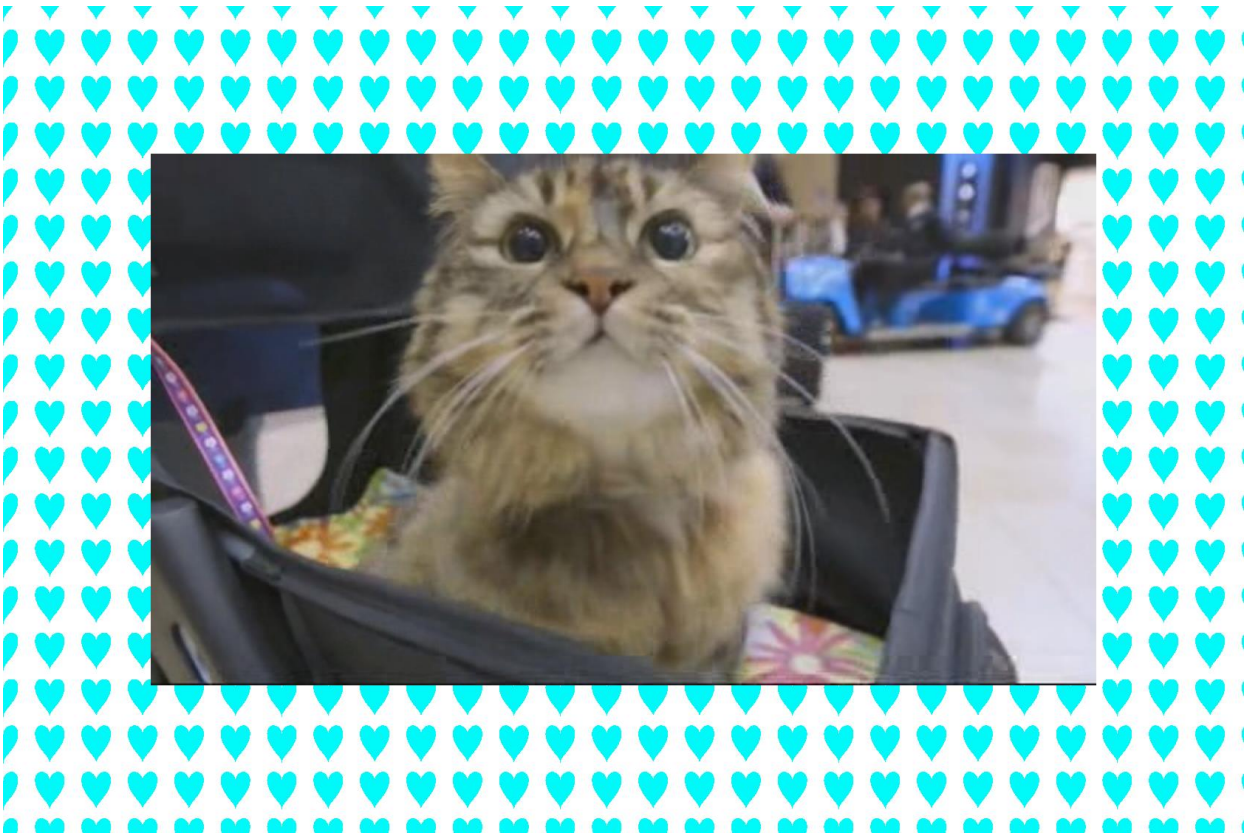
```
./main ./kitten.bmp -d -p 1 -w 80 -F 250.200.0 -o kitten_output.bmp
```



Второй узор (pattern = 2):



Третий узор (pattern = 3):



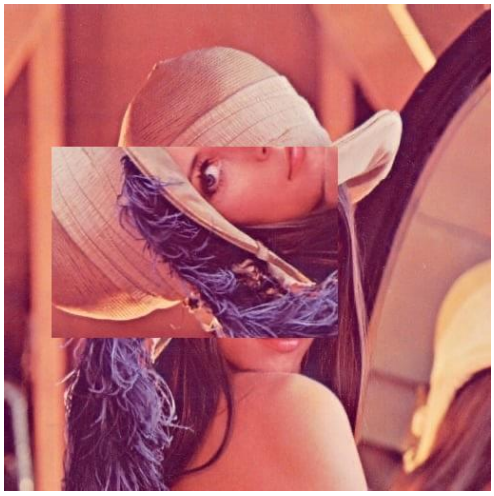
5. Пример выполнения индивидуального задания №3 — Поворот изображения:

Угол 90 градусов:

```
./main ./woman.bmp -t -f 100.100 -s 300.400 -a 90 -o woman_output.bmp
```

Угол 180 градусов:

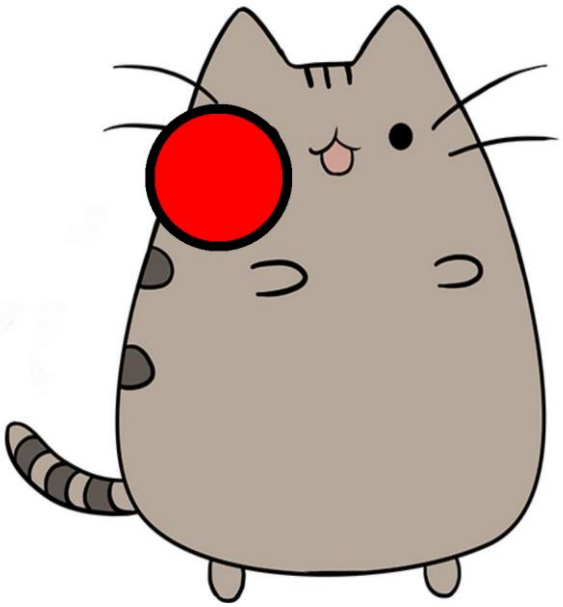
Угол 270 градусов:



6. Пример выполнения индивидуального задания №4 — Рисование окружности:

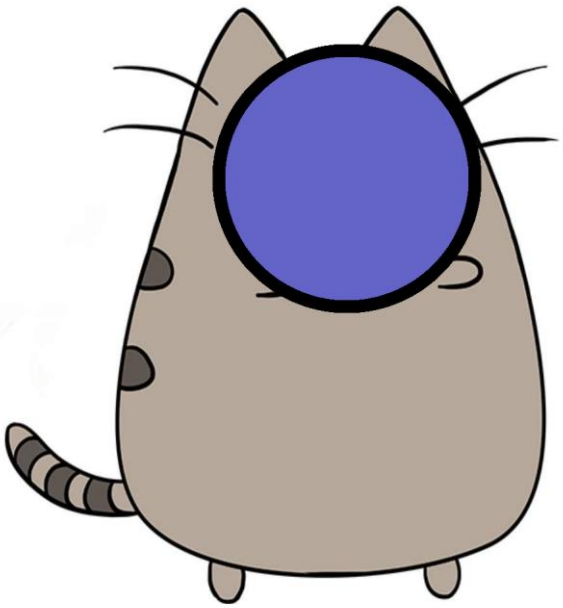
Задание окружности с помощью радиуса и центра окружности:

```
./main ./kitten.bmp -c -1 100.300.400 -w 15 -F 0.0.0 -H 1 -S 255.0.0 -o kitten_output.bmp
```



Задание окружности с помощью левой верхней и правой нижней точек описываемого квадрата:

```
$ ./main ./kitten.bmp -c -2 100.400.500.900 -w 10 -H 1 -F 0.0.0 -S 100.100.200 -o output_kitten
```



ПРИЛОЖЕНИЕ В
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл: main.c

```
#include "cli.h"


int main(int argc, char** argv){
    cli(argc, argv);
    return 0;
}
```

Файл: cw_functions.c

```
#include <stdlib.h>
#include <math.h>
#include "print.h"
#include "cw_functions.h"
#include "bmp_processing.h"

// закрашивание точки
void draw_point(RGB *elem, unsigned char r, unsigned char g, unsigned char b) {
```

```
    elem->r = r;
    elem->g = g;
    elem->b = b;
}

// 1 функция - рисование прямоугольника
BMP_file* rectangle(BMP_file * bmp_file, Arguments *arguments) {
    int *x1 = &(arguments->x1), *y1 = &(arguments->y1);
    int *x2 = &(arguments->x2), *y2 = &(arguments->y2);
    int *line_wight = &(arguments->line_width);
    int *line_color_r = &(arguments->color1_r);
    int *line_color_g = &(arguments->color1_g);
    int *line_color_b = &(arguments->color1_b);
    int *shape_fill = &(arguments->shape);
    int *color_fill_r = &(arguments->color2_r);
    int *color_fill_g = &(arguments->color2_g);
    int *color_fill_b = &(arguments->color2_b);

    if (check_color(*line_color_r) || check_color(*line_color_g) ||
check_color(*line_color_b)){
        err_color();
    }
    if (check_color(*color_fill_r) || check_color(*color_fill_g) ||
check_color(*color_fill_b)){
        err_color();
    }
    if (*x1 < 0 || *y1 < 0 || *x2 >= bmp_file->dhdr.width || *y2 >=
bmp_file->dhdr.height) {
        err_limit_img();
    }
    if(*line_wight < 0 ||  (*shape_fill != 0 && *shape_fill != 1)) {
        err_value();
    }

    for (int i = 0; i < *line_wight; i++) {
        for (int j = *x1; j <= *x2; j++) {
            // Верхняя граница
            draw_point(&bmp_file->pixels_arr[*y1+i][j], *line_color_r, *line_color_g,
*line_color_b);
            // Нижняя граница
            draw_point(&bmp_file->pixels_arr[*y2 - i][j], *line_color_r, *line_color_g,
*line_color_b);
        }
        for (int j = *y1; j <= *y2; j++) {
```

```
            // Левая граница
            draw_point(&bmp_file->pixels_arr[j][*x1 + i], *line_color_r, *line_color_g,
*line_color_b);
            // Правая граница
            draw_point(&bmp_file->pixels_arr[j][*x2 - i], *line_color_r, *line_color_g,
*line_color_b);
        }
    }

//     закраска прямоугольника, если shape_fill = 1
    if (*shape_fill == 1) {
        for (int i = *y1 + *line_wight; i <= *y2 - *line_wight; i++) {
            for (int j = *x1 + *line_wight; j <= *x2 - *line_wight; j++) {
                draw_point(&bmp_file->pixels_arr[i][j], *color_fill_r, *color_fill_g,
*color_fill_b);
            }
        }
    }
    *x1 = -1, *y1 = -1, *x2 = -1, *y2 = -1, *line_wight = -1, * shape_fill = -1;
    *line_color_r = -1, *line_color_g = -1, *line_color_b = -1;
    *color_fill_r = -1, *color_fill_g = -1, *color_fill_b = -1;

    return bmp_file;
}

// 2 функция - поворот изображения
BMP_file * turn(BMP_file * bmp_file, Arguments *arguments) {
    int *x1 = &(arguments->x1), *y1 = &(arguments->y1);
    int *x2 = &(arguments->x2), *y2 = &(arguments->y2);
    int *angle = &(arguments->angle);
    if (*x1 < 0 || *y1 < 0 || *x2 >= bmp_file->dhdr.width || *y2 >=
bmp_file->dhdr.height) {
        err_limit_img();
        exit(-1);
    }
    if (*angle % 90 != 0 || *angle > 270 || *angle < 0) {
        err_value();
    }
    RGB **tmp_arr = malloc((bmp_file->dhdr.height + 1) * sizeof(RGB *));
    for (int i = 0; i < bmp_file->dhdr.height; i++) {
        tmp_arr[i] = calloc((bmp_file->dhdr.width + 1), sizeof(RGB));
    }
    if (*angle == 90) {
        if ((*x2 - *x1) >= (*y2 - *y1)) {
```

```
    int t_h = ((*x2 - *x1) - (*y2 - *y1)) / 2 ;
    int t_h1 = t_h;
    int t_h2 = t_h;
    if (t_h >= *y1) {
        t_h1 = *y1;
    }
    if (t_h >= bmp_file->dhdr.height - *y2) {
        t_h2 = bmp_file->dhdr.height - *y2 - 1;
    }
    int tx = 0;
    int ty = 0;
    for (int j = *y1; j <= *y2; j++) {
        tx = 0;
        for (int i = *x1 + t_h - t_h1; i <= *x2 - t_h + t_h2; i++) {
            tmp_arr[ty][tx] = bmp_file->pixels_arr[j][i];
            tx++;
        }
        ty++;
    }
    tx = 0;
    ty = 0;
    for (int j = *x2 - t_h; j > *x1 + t_h; j--) {
        for (int i = *y1 - t_h1; i < *y2 + t_h2; i++) {
            bmp_file->pixels_arr[i][j] = tmp_arr[ty][tx];
            tx++;
        }
        tx = 0;
        ty++;
    }
} else {
    int t_h = ((*y2 - *y1) - (*x2 - *x1)) / 2;
    int t_h1 = t_h;
    int t_h2 = t_h;
    if (t_h >= *x1) {
        t_h2 = *x1;
    }
    if (t_h > bmp_file->dhdr.width - *x2) {
        t_h1 = bmp_file->dhdr.width - *x2 - 1;
    }
    int tx = 0;
    int ty = 0;
    for (int j = *y1 + t_h - t_h1; j <= *y2 - t_h + t_h2; j++) {
        tx = 0;
        for (int i = *x1; i <= *x2; i++) {
```

```
                tmp_arr[ty][tx] = bmp_file->pixels_arr[j][i];
                tx++;
            }
            ty++;
        }
        tx = 0;
        ty = 0;
        for (int i = *x2 + t_h1; i >= *x1 - t_h2; i--) {
            tx = 0;
            for (int j = *y1 + t_h; j <= *y2 - t_h; j++) {
                bmp_file->pixels_arr[j][i] = tmp_arr[ty][tx];
                tx++;
            }
            ty++;
        }
    }
}
if (*angle == 180) {
    int tx = 0;
    int ty = 0;
    for (int j = *y1; j <= *y2; j++) {
        for (int i = *x1; i <= *x2; i++) {
            tmp_arr[ty][tx] = bmp_file->pixels_arr[j][i];
            tx++;
        }
        tx = 0;
        ty++;
    }
    tx = 0;
    ty = 0;
    for (int j = *y2; j >= *y1; j--) {
        for (int i = *x2; i >= *x1; i--) {
            bmp_file->pixels_arr[j][i] = tmp_arr[ty][tx];
            tx++;
        }
        tx = 0;
        ty++;
    }
}
if (*angle == 270) {
    if ((*x2 - *x1) > (*y2 - *y1)) {
        int t_h = ((*x2 - *x1) - (*y2 - *y1)) / 2;
        int t_h1 = t_h;
        int t_h2 = t_h;
```

```
    if (t_h > *y1) {
        t_h1 = *y1;
    }
    if (t_h > bmp_file->dhdr.height - *y2) {
        t_h2 = bmp_file->dhdr.height - *y2 - 1;
    }
    int tx = 0;
    int ty = 0;
    for (int j = *y1; j <= *y2; j++) {
        for (int i = *x1 + t_h - t_h2; i <= *x2 - t_h + t_h1; i++) {
            tmp_arr[ty][tx] = bmp_file->pixels_arr[j][i];
            tx++;
        }
        tx = 0;
        ty++;
    }
    tx = 0;
    ty = 0;
    for (int i = *x1 + t_h; i < *x2 - t_h; i++) {
        for (int j = *y2 + t_h2; j > *y1 - t_h1; j--) {
            bmp_file->pixels_arr[j][i] = tmp_arr[ty][tx];
            tx++;
        }
        tx = 0;
        ty++;
    }
} else {
    int t_h = ((*y2 - *y1) - (*x2 - *x1)) / 2 - ((*y2 - *y1) - (*x2 - *x1)) % 2;
    int t_h1 = t_h;
    int t_h2 = t_h;
    if (t_h > *x1) {
        t_h1 = *x1;
    }
    if (t_h > bmp_file->dhdr.width - *x2) {
        t_h2 = bmp_file->dhdr.width - *x2 - 1;
    }
    int tx = 0;
    int ty = 0;
    for (int j = *y1 + t_h - t_h1; j <= *y2 - t_h + t_h2; j++) {
        for (int i = *x1; i <= *x2; i++) {
            tmp_arr[ty][tx] = bmp_file->pixels_arr[j][i];
            tx++;
        }
        tx = 0;
```

```
            ty++;
        }
        tx = 0;
        ty = 0;
        for (int i = *x1 - t_h1; i < *x2 + t_h2; i++) {
            for (int j = *y2 - t_h; j > *y1 + t_h; j--) {
                bmp_file->pixels_arr[j][i] = tmp_arr[ty][tx];
                tx++;
            }
            tx = 0;
            ty++;
        }
    }
  }
  for (int i = 0; i < bmp_file->dhdr.height; i++) {
    free(tmp_arr[i]);
  }
  free(tmp_arr);

  *x1 = -1, *y1 = -1, *x2 = -1, *y2 = -1, *angle = -1;
  return bmp_file;
}

void frame_pattern_1(int frame_width, int frame_r, int frame_g, int frame_b,
BMP_file* new_bmp_file){
  for (int i = 0; i < new_bmp_file->dhdr.height; i++){
    for (int j = 0; j < new_bmp_file->dhdr.width; j++){
      if ( sin(6.0 * i / frame_width) < sin(6.0 * j / frame_width) ){
        draw_point(&new_bmp_file->pixels_arr[i][j], frame_r, frame_g, frame_b);
      } else{
        draw_point(&new_bmp_file->pixels_arr[i][j], 255, 255,255);
      }
    }
  }
}

void frame_pattern_2(int frame_width, int frame_r, int frame_g, int frame_b,
BMP_file* new_bmp_file){
  for (int i = 0; i < new_bmp_file->dhdr.height; i++){
    for (int j = 0; j < new_bmp_file->dhdr.width; j++){
      if ( tan(60.0/frame_width * i/10.0 - sin(60.0/frame_width * j/10.0)) <=
sin(60.0/frame_width * j/10.0)){
        draw_point(&new_bmp_file->pixels_arr[i][j], frame_r, frame_g, frame_b);
      } else{
```

```
            draw_point(&new_bmp_file->pixels_arr[i][j], 255, 255,255);
        }
    }
  }
}
void frame_pattern_3(int frame_width, int frame_r, int frame_g, int frame_b,
BMP_file* new_bmp_file){
  for (int i = 0; i < new_bmp_file->dhdr.height; i++){
    for (int j = 0; j < new_bmp_file->dhdr.width; j++){
      double a = atan(tan(57.0/frame_width * j/10.0));
      double b = atan(tan(-57.0/frame_width * i/10.0));
      if ( (a*a + (b - sqrt(fabs(a))) * (b - sqrt(fabs(a)))) <= 1  ){
        draw_point(&new_bmp_file->pixels_arr[i][j], frame_r, frame_g, frame_b);
      } else{
        draw_point(&new_bmp_file->pixels_arr[i][j], 255, 255,255);
      }
    }
  }
}
// 3 функция - рамка
BMP_file * draw_frame(BMP_file *bmp_file, Arguments *arguments) {
  int *pattern = &(arguments->pattern);
  int *frame_width = &(arguments->line_width);
  int *color_r = &(arguments->color1_r), *color_g = &(arguments->color1_g),
*color_b = &(arguments->color1_b);
  if (check_color(*color_r) || check_color(*color_g) || check_color(*color_b) ||
*frame_width < 0){
    err_value();
  }
  // новое изображение
  BMP_file* new_bmp_file;
  new_bmp_file = new_bmp(bmp_file->dhdr.width + 2*(*frame_width),
bmp_file->dhdr.height + 2*(*frame_width), bmp_file);
//   fprintf(stderr, "error");

  // рамка на новом изображении
  switch (*pattern) {
    case 1:
      frame_pattern_1(*frame_width, *color_r, *color_b, *color_g, new_bmp_file);
      break;
    case 2:
      frame_pattern_2(*frame_width, *color_r, *color_b, *color_g, new_bmp_file);
      break;
    case 3:
```

```
            frame_pattern_3(*frame_width, *color_r, *color_b, *color_g, new_bmp_file);
            break;
        default:
            err_value();
            break;
    }
    // отрисовка старого
    for (int i = 0; i < bmp_file->dhdr.height; i++){
        for (int j = 0; j < bmp_file->dhdr.width; j++){
            RGB* pixel = &bmp_file->pixels_arr[i][j];
            draw_point(&new_bmp_file->pixels_arr[i + *frame_width][j +
*frame_width], pixel->r, pixel->g, pixel->b);
        }
    }
    free_bmp_file(bmp_file);
    *frame_width = -1,   *color_r = -1, *color_g = -1, *color_b = -1;
    return new_bmp_file;
}

// 4 функция - окружность
BMP_file * circle(BMP_file * bmp_file, Arguments *arguments) {

    int *radius = &(arguments->radius);
    int *x_center = &(arguments->x_c), *y_center = &(arguments->y_c);
    int *x1 = &(arguments->x1), *y1 = &(arguments->y1);
    int *x2 = &(arguments->x2), *y2 = &(arguments->y2);
    int *line_wight = &(arguments->line_width);
    int *line_color_r = &(arguments->color1_r);
    int *line_color_g = &(arguments->color1_g);
    int *line_color_b = &(arguments->color1_b);
    int *shape_fill = &(arguments->shape);
    int *color_fill_r = &(arguments->color2_r);
    int *color_fill_g = &(arguments->color2_g);
    int *color_fill_b = &(arguments->color2_b);

    // проверка  коррекности введенных значений
    if ( (*radius != -1 && *radius < 0) || (*x_center != -1 && *x_center < 0) ||
(*y_center != -1 && *y_center < 0)){
        err_value();
    }
    if ( (*x1 != -1 && *x1 < 0) || (*x2 != -1 && *x2 < 0)){
        err_value();
    }
    if ( (*y1 != -1 && *y1 < 0) || (*y2 != -1 && *y2 < 0) ){
```

```
        err_value();
    }
    if ((*shape_fill != 0 && *shape_fill != 1) || *line_wight < 0){
        err_value();
    }
    if (check_color(*line_color_r) || check_color(*line_color_g) ||
check_color(*line_color_b)){
        err_color();
    }
    if (check_color(*color_fill_r) || check_color(*color_fill_g) ||
check_color(*color_fill_b)){
        err_color();
    }
    // проверка выхода за картинку
    if ((*x2 > 0 && *x2 >= bmp_file->dhdr.width) || (*y2 > 0 && *y2 >=
bmp_file->dhdr.width)){
        err_limit_img();
    }
    int sum_radius_x = (*radius + *line_wight );
    if ( (*radius > 0) && (sum_radius_x > *x_center || sum_radius_x > *y_center ||
sum_radius_x > bmp_file->dhdr.width - *x_center || sum_radius_x >
bmp_file->dhdr.height - *y_center ) ){
        err_limit_img();
    }
    // 2 способ - через две точки
    if (*radius == -1){
        *radius = (*x2 - *x1)/2 - *line_wight;
        if (*radius < 0){
            err_value();
        }
        *x_center = *x1 + *radius + *line_wight;
        *y_center = *y1 + *radius + *line_wight;
        for (int y = 0; y < bmp_file->dhdr.height; y++){
            for (int x = 0; x < bmp_file->dhdr.width; x++){
                int dx = x - *x_center;
                int dy = y - *y_center;
                int hypotenuse = dx*dx + dy*dy;
                if (hypotenuse < (*radius) * (*radius) && *shape_fill == 1 ){
                    draw_point(&bmp_file->pixels_arr[x][y], *color_fill_r, *color_fill_g,
*color_fill_b);
                }
                if (hypotenuse >= (*radius)*(*radius) && hypotenuse < (*radius + 2
*(*line_wight))*(*radius + 2*(*line_wight))){
```

```
                draw_point(&bmp_file->pixels_arr[x][y], *line_color_r, *line_color_g,
*line_color_b);
            }
        }
    }
  }
  else{
    for (int y = 0; y < bmp_file->dhdr.height; y++){
      for (int x = 0; x < bmp_file->dhdr.width; x++){
        int dx = x - *x_center;
        int dy = y - *y_center;
        int hypotenuse = dx*dx + dy*dy;
        if (hypotenuse < (*radius) * (*radius) && *shape_fill == 1 ){
          draw_point(&bmp_file->pixels_arr[x][y], *color_fill_r, *color_fill_g,
*color_fill_b);
        }
        if (hypotenuse >= (*radius)*(*radius) && hypotenuse < (*radius +
*line_wight)*(*radius + *line_wight)){
          draw_point(&bmp_file->pixels_arr[x][y], *line_color_r, *line_color_g,
*line_color_b);
        }
      }
    }
  }

  *x1 = -1, *y1 = -1, *x2 = -1, *y2 = -1, *line_wight = -1, * shape_fill = -1;
  *line_color_r = -1, *line_color_g = -1, *line_color_b = -1;
  *color_fill_r = -1, *color_fill_g = -1, *color_fill_b = -1;
  *radius = -1; *x_center = -1; *y_center = -1;

  return bmp_file;
}
```

Файл: cw_all_functions .h

```
#pragma once
#include "bmp_processing.h"

void draw_point(RGB *elem, unsigned char r, unsigned char g, unsigned char b);
BMP_file * rectangle(BMP_file * bmp_file, Arguments *arguments);
BMP_file * draw_frame(BMP_file * bmp_file, Arguments *arguments);
BMP_file * turn(BMP_file * bmp_file, Arguments *arguments);
BMP_file * circle(BMP_file * bmp_file, Arguments *arguments);
```

Файл: cli.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>
#include "cw_functions.h"
#include "bmp_processing.h"
#include "print.h"

#pragma pack (push,1)

const struct option long_options[] = {
        {"g_rectangle",       no_argument,       NULL, 'g'},
        {"draw_frame",      no_argument,       NULL, 'd'},
        {"turn",          no_argument,       NULL, 't'},
        {"circle",        no_argument,       NULL, 'c'},
        {"first",         required_argument, NULL, 'f'},
        {"second",        required_argument, NULL, 's'},
        {"angle",         required_argument, NULL, 'a'},
        {"path_1", required_argument, NULL, '1'},
        {"path_2",        required_argument, NULL, '2'},
        {"width",         required_argument, NULL, 'w'},
        {"pattern_frame",         required_argument, NULL, 'p'},
        {"line_color",   required_argument, NULL, 'F'},
        {"shape_color",  required_argument, NULL, 'S'},
        {"shape", required_argument, NULL, 'H'},
        {"output",        required_argument, NULL, 'o'},
        {"help",          no_argument,       NULL, 'h'},
        {"info",          no_argument,       NULL, 'i'},
        {NULL,            no_argument,       NULL,  0 }
};

#pragma pack(pop)

void parse(int **arr, int count, char *opt_arg) {
    char *str;
    str = strtok(opt_arg, ".");
    *(arr[0]) = atoi(str);
    for(int i = 1; i < count; i++) {
        str = strtok(NULL, ".");
        if(str != NULL) *(arr[i]) = atoi(str);
        else {
            printf("Было введено недостаточно чисел в одном аргументе. Работа
программы завершена.\n");
            exit(-1);
```

```
        }
    }
}


Arguments* choice(Arguments *arguments, int opt) {
    int **arr = malloc(5 * sizeof(int*));

    switch (opt) {
        case 'f':
            arr[0] = &(arguments->x1), arr[1] = &(arguments->y1);
            parse(arr, 2, optarg);
            break;
        case 's':
            arr[0] = &(arguments->x2), arr[1] = &(arguments->y2);
            parse(arr, 2, optarg);
            break;
        case 'a':
            arguments->angle = atoi(optarg);
            break;
        case 'p':
            arguments->pattern = atoi(optarg);
            break;
        case '1':
            arr[0] = &(arguments->radius), arr[1] = &(arguments->x_c), arr[2] =
&(arguments->y_c);
            parse(arr, 3, optarg);
            break;
        case '2':
            arr[0] = &(arguments->x1), arr[1] = &(arguments->y1);
            arr[2] = &(arguments->x2), arr[3] = &(arguments->y2);
            parse(arr, 4, optarg);
            break;
        case 'w':
            arguments->line_width = atoi(optarg);
            break;
        case 'F':
            arr[0] = &(arguments->color1_r), arr[1] = &(arguments->color1_g), arr[2] =
&(arguments->color1_b);
            parse(arr, 3, optarg);
            break;
        case 'S':
            arr[0] = &(arguments->color2_r), arr[1] = &(arguments->color2_g), arr[2] =
&(arguments->color2_b);
```

```
            parse(arr, 3, optarg);
            break;
        case 'H':
            arguments->shape = atoi(optarg);
            break;
        case 'o':
            arguments->output = malloc(sizeof(optarg)+1);
            strcpy(arguments->output, optarg);
            break;
        case 'h':
            printHelp();
            break;
    };

    return arguments;
}

BMP_file * functions_choice(int opt, int prev_opt, Arguments *arguments, BMP_file
* bmp_file) {
    if (opt == 'g' || opt == 'd' || opt == 't' || opt == 'c') {
        switch (prev_opt) {
            case 'g':
                bmp_file = rectangle(bmp_file, arguments);
                break;
            case 'd':
                bmp_file = draw_frame(bmp_file, arguments);
                break;
            case 't':
                bmp_file = turn(bmp_file, arguments);
                break;
            case 'c':
                bmp_file = circle(bmp_file, arguments);
                break;
        }
    }
    return bmp_file;
}

Arguments* arg_set_to_NULL(Arguments* arguments) {
    arguments->x1 = -1, arguments->x2 = -1, arguments->x_c = -1;
    arguments->y1 = -1, arguments->y2 = -1, arguments->y_c = -1;
    arguments->angle = -1, arguments->line_width = -1, arguments->output = NULL;
    arguments->color1_r = -1, arguments->color1_g = -1, arguments->color1_b = -1;
    arguments->color2_r = -1, arguments->color2_g = -1, arguments->color2_b = -1;
```

```
    arguments->shape = -1, arguments->radius = -1; arguments->pattern = -1;
    return arguments;
}


void cli(int argc, char **argv) {
    char *opts = "gdtcf:s:1:a:2:w:F:H:S:p:o:m:hi";
    Arguments *arguments = malloc(sizeof(Arguments));
    arguments = arg_set_to_NULL(arguments);

    int opt;
    int prev_opt = 'N';
    int long_index;
    char* file_name = argv[1];

    if (argc == 1) {
        printHelp();
        exit(-1);
    }

    BMP_file * bmp_file = read_bmp(file_name);

    arguments->output = file_name;

    opt = getopt_long(argc, argv, opts, long_options, &long_index);
    while (opt != -1) {

        bmp_file = functions_choice(opt, prev_opt, arguments, bmp_file);

        if (opt == 'g' || opt == 'd' || opt == 't' || opt == 'c'){
            prev_opt = opt;
        }
        if (opt == 'i') {
            print_bmp_info(bmp_file);
        }

        arguments = choice(arguments, opt);

        opt = getopt_long(argc, argv, opts, long_options, &long_index);

    }

    bmp_file = functions_choice('g', prev_opt, arguments, bmp_file);
```

```
//    printf("wight = %d, height = %d\n", bmp_file->dhdr.width,
bmp_file->dhdr.height);
//    printf("x1 = %d; y1 = %d; x2 = %d; y2 = %d; x_c = %d; y_c = %d, pattern
= %d\n", arguments->x1, arguments->y1, arguments->x2, arguments->y2,
arguments->x_c, arguments->y_c, arguments->pattern);
//    printf("angle = %d, line_width = %d, radius = %d\n", arguments->angle,
arguments->line_width, arguments->radius);
//    printf("r1 = %d, g1 = %d, b1 = %d, r2 = %d, g2 = %d, b2 = %d\n",
arguments->color1_r, arguments->color1_g, arguments->color1_b,
arguments->color2_r, arguments->color2_g, arguments->color2_b);
//    printf("output = %s\n", arguments->output);
//    printf("file_name = %s\n",file_name);
//    printf("argc = %d\n",argc);

    output_bmp_file(bmp_file, arguments->output);
    free_bmp_file(bmp_file);
}
```

Файл: cli.h

```
#pragma once
#include "bmp_processing.h"

void printHelp();
void parse(int **arr, int count, char *opt_arg);
Arguments* choice(Arguments *arguments, int opt);
void cli(int argc, char **argv);
```

Файл*: bmp_processing.c*

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "bmp_processing.h"
#include "print.h"

// заполняем pixels_arr
void fill_img(FILE* fp, BMP_file** bmp_file) {
    int bytes_per_pixel = (*bmp_file)->dhdr.bits_per_pixel / 8; // количество байт на
пиксель
    int row_size = bytes_per_pixel * (*bmp_file)->dhdr.width; // размер одной
строки в байтах
    int row_padding = (4 - (row_size % 4)) % 4; // количество добавочных байт для
выравнивания
    // последнее действие, если количество байт и так кратно 4
```

```
    int cur_col;
    int height = (*bmp_file)->dhdr.height - 1;
    unsigned char *row = malloc(row_size + row_padding); // место для считывания строки полностью (+выравнивание)
    fseek(fp, (*bmp_file)->bhdr.pixel_offset, SEEK_SET);
    // точка смещения: SEEK_SET – смещение отсчитывается от начала файла
    for (int cur_row = 0; cur_row < height + 1; cur_row++) {
        cur_col = 0;
        fread(row, row_size+row_padding, 1, fp); // считывание данных из потока
        for (int i = 0; i < row_size; i += bytes_per_pixel){ // проходимся по пикселям в строке
            (*bmp_file)->pixels_arr[height - cur_row][cur_col].b = row[i];
            (*bmp_file)->pixels_arr[height - cur_row][cur_col].g = row[i+1];
            (*bmp_file)->pixels_arr[height - cur_row][cur_col].r = row[i+2];
            cur_col++;
        }
    }
    free(row);
}

BMP_file* read_bmp(char* f_name){
    FILE* fp = fopen(f_name, "r");
    if (!fp){
        err_find_file();
    }
    BMP_file* bmp_file = malloc(sizeof (BMP_file));
    fread(&bmp_file->bhdr, sizeof (BMPHeader ), 1, fp);
    fread(&bmp_file->dhdr, sizeof (DIBHeader), 1, fp);

    // проверка на расширение файла
    if (strlen(f_name) > 4 && strcmp(f_name + strlen(f_name) - 4, ".bmp")){
        err_file_extension();
    }
    // проверка на весию файла
    if (bmp_file->dhdr.header_size != 40){
        err_version();
    }
    // проверка на количество бит
    if (bmp_file->dhdr.bits_per_pixel != 24) {
        count_bits();
    }

    bmp_file->pixels_arr = malloc(bmp_file->dhdr.height * sizeof(RGB*));
    for(int i = 0; i < bmp_file->dhdr.height; i++) {
```

```
        bmp_file->pixels_arr[i] = malloc(bmp_file->dhdr.width * sizeof(RGB));
    }
    fill_img(fp, &bmp_file);
    fclose(fp);

    return bmp_file;
}

unsigned char* rgb_to_str(BMP_file* bmp_file) {
    int bytes_per_pixel = bmp_file->dhdr.bits_per_pixel / 8 ;
    int row_size = bytes_per_pixel * bmp_file->dhdr.width;
    int row_padding = (4 - (row_size % 4)) % 4;
    int height = bmp_file->dhdr.height;
    int width = bmp_file->dhdr.width;

    unsigned char *data = malloc((row_size + row_padding) * bmp_file->dhdr.height);
    int index = 0;
    for(int i = 0; i < height; i++) {
        for(int j = 0; j < width; j++) {
            data[index++] = bmp_file->pixels_arr[height - i - 1][j].b;
            data[index++] = bmp_file->pixels_arr[height - i - 1][j].g;
            data[index++] = bmp_file->pixels_arr[height - i - 1][j].r;
        }
        for(int j = 0; j < row_padding; j++) {
            data[index++] = (unsigned char)0;
        }
    }
    return data;
}

void output_bmp_file(BMP_file * bmp_file, char* output_file) {
    FILE *fp = fopen(output_file, "wb"); // "wb" - создает двоичный файл для
записи.
    if (!fp) {
        printf("Проблемы с созданием файла для вывода. Работа программы
завершена\n");
        exit(-1);
    }
    fwrite(&bmp_file->bhdr, 1, sizeof(BMPHeader),fp);
    fwrite(&bmp_file->dhdr, 1, sizeof(DIBHeader),fp);

    unsigned char* data = rgb_to_str(bmp_file);

    int bytes_per_pixel = bmp_file->dhdr.bits_per_pixel / 8 ;
```

```
    int row_size = bytes_per_pixel * bmp_file->dhdr.width;
    int row_padding = (4 - (row_size % 4)) % 4;

    fwrite(data, 1, (row_size + row_padding) * bmp_file->dhdr.height, fp);
    // записывает до count эл из буффера в fp
    free(data);
    fclose(fp);
}
BMP_file * new_bmp(int wight_f, int height_f, BMP_file* bmp_file) {
    BMP_file* new_file = calloc(1, sizeof (BMP_file));
    new_file->dhdr.width = wight_f;
    new_file->dhdr.height = height_f;
    new_file->bhdr.file_size = sizeof (RGB) * wight_f * height_f + sizeof
(BMPHeader) + sizeof (DIBHeader);
    new_file->dhdr.data_size = sizeof (RGB) * wight_f * height_f;

    new_file->bhdr.ID[0] = bmp_file->bhdr.ID[0];
    new_file->bhdr.ID[1] = bmp_file->bhdr.ID[1];
    new_file->bhdr.pixel_offset = bmp_file->bhdr.pixel_offset;
    new_file->bhdr.unused[0] = bmp_file->bhdr.unused[0];
    new_file->bhdr.unused[1] = bmp_file->bhdr.unused[1];
    new_file->bhdr.unused[2] = bmp_file->bhdr.unused[2];
    new_file->bhdr.unused[3] = bmp_file->bhdr.unused[3];

    new_file->dhdr.header_size = bmp_file->dhdr.header_size;
    new_file->dhdr.planes = bmp_file->dhdr.planes;
    new_file->dhdr.bits_per_pixel = bmp_file->dhdr.bits_per_pixel;
    new_file->dhdr.compression = bmp_file->dhdr.compression;
    new_file->dhdr.x_pixels_per_meter = bmp_file->dhdr.x_pixels_per_meter;
    new_file->dhdr.y_pixels_per_meter = bmp_file->dhdr.y_pixels_per_meter;
    new_file->dhdr.colors_in_color_table = bmp_file->dhdr.colors_in_color_table;
    new_file->dhdr.important_color_count = bmp_file->dhdr.important_color_count;
    new_file->bhdr.pixel_offset = bmp_file->bhdr.pixel_offset;
    new_file->pixels_arr = malloc(height_f * sizeof(RGB*));
    for(int i = 0; i < height_f; i++) {
        new_file->pixels_arr[i] = calloc(wight_f,sizeof(RGB));
    }
    return new_file;
}
void free_bmp_file(BMP_file* bmp_file) {
    if (bmp_file) {
        for(int i = 0; i < bmp_file->dhdr.height; i++) {
            free(bmp_file->pixels_arr[i]);
        }
```

```
        free(bmp_file->pixels_arr);
        free(bmp_file);
    }
}
```

Файл: *bmp_processing.h*

```
#pragma once
#include <stdio.h>
#include <getopt.h>
#pragma pack (push, 1)

typedef struct Arguments {
    int x1, y1;
    int x2, y2;
    int x_c, y_c;
    int radius;
    int angle;
    int shape;
    int pattern;
    int line_width;
    int color1_r, color1_g, color1_b;
    int color2_r, color2_g, color2_b;
    char *output;
} Arguments;

typedef struct BMPHeader{
    unsigned char ID[2];
    unsigned int file_size;
    unsigned char unused[4];
    unsigned int pixel_offset;
} BMPHeader;

typedef struct DIBHeader{
    unsigned int header_size;
    unsigned int width;
    unsigned int height;
    unsigned short planes; // слои
    unsigned short bits_per_pixel; // бит на пиксель
    unsigned int compression; // сжатие
    unsigned int data_size; // размер информации о пикселях
    unsigned int x_pixels_per_meter;
    unsigned int y_pixels_per_meter;
    unsigned int colors_in_color_table; // количество цветов в палитре
    unsigned int important_color_count; // количество важных цветов в палитре
```

```
} DIBHeader;

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

typedef struct BMP_file{
    BMPHeader bhdr;
    DIBHeader dhdr;
    unsigned char *data;
    RGB **pixels_arr; // память для набора пикселей (цвет)
} BMP_file;

#pragma pack(pop)

void fill_img(FILE* fp, BMP_file** bmp_file);
BMP_file* read_bmp(char* f_name);
void free_bmp_file(BMP_file* bmp_file);
unsigned char* rgb_to_str(BMP_file* bmp_file);
void output_bmp_file(BMP_file * bmp_file, char* output_file);
BMP_file * new_bmp(int wight, int height, BMP_file* bmp_file);
```

Файл: print.c

```
#include "print.h"
#include <stdio.h>
#include <stdlib.h>

void err_limit_img(){
    printf("Указанные координаты не находятся в пределах изображения.\n");
    exit(-1);

}
void err_value(){
    printf("Указано неверное значение переменной.\n");
    exit(-1);

}
int check_color(int color) {
    return !(0 <= color && color <= 255);
}
void err_color(){
    printf("Указано неверное значение цвета.\n");
```

```
    exit(-1);
}
void err_file_extension(){
    printf("Программа работает только с файлами, у которых расширение .bmp.
Работа программы завершена.\n");
    exit(-1);
};
void err_version(){
    printf("Программа работает только с фйлами 3 версии BMP. Работа
программы завершена\n");
    exit(-1);
}
void err_find_file(){
    printf("Не удается загрузить файл.\n");
    exit(-1);
}
void count_bits(){
    printf("Программа работает только с файлами, у которых выделено 24 бита на
цвет. Работа программы завершена.\n");
    exit(-1);
}
void print_bmp_pixels(BMP_file *bmp_file){
    for (int i = 0; i<bmp_file->dhdr.height; i++){
        for (int j = 0; j<bmp_file->dhdr.width; j++){
            printf("%02x %02x %02x ", bmp_file->pixels_arr[i][j].r,
bmp_file->pixels_arr[i][j].g, bmp_file->pixels_arr[i][j].b);
        }
    }
    printf("\n");
}

void printHelp() {
    printf("Руководство по использованию программы:\n");
    printf("Программа обрабатывает BMP-файлы версии [...]. С глубиной
изображения 24 бита на пиксель\n");
    printf("Формат ввода: <имя исполняемого файла> <имя BMP-файла для
обработки> <функция> -<ключ>/--<полный ключ> <аргумент> ... \n");
    printf("Функции/ключи:\n");

    printf("Первый ключ: -g/--g_rectangle-- Рисование прямоугольника.\n");
    printf("\t-f/--first  [<x-координата>.<y-координата>] - координата левого
верхнего угла фигуры.\n");
    printf("\t-s/--second [<x-координата>.<y-координата>] - координата правого
нижнего угла фигуры.\n");
```

```
    printf("\t-w/--width  [<число>] - толщина линии\n");
    printf("\t-F/--line_color  [<число>.<число>.<число>] - цвет линии (RGB).\n");
    printf("\t-H/--shape  [<число>] - выбор наличия заливки фигуры (0 - не залит, 1
- залит).\n");
    printf("\t-S/--shape_color [<число>.<число>.<число>] - цвет заливки
(RGB).\n");

    printf("Второй ключ: -d/--draw_frame -- Рисование рамки в виде узора.\n");
    printf("\t-p/--pattern [<число>] - номер узора рамки, число от 1 до 3.\n");
    printf("\t-F/--line_color  [<число>.<число>.<число>] - цвет рамки(RGB).\n");
    printf("\t-w/--width  [<число>] - ширина рамки.\n");

    printf("Третий ключ: -t/--turn -- Поворот изображения (части) на 90/180/270
градусов.\n");
    printf("\t-f/--first  [<x-координата>.<y-координата>] - координата левого
верхнего угла фигуры.\n");
    printf("\t-s/--second [<x-координата>.<y-координата>] - координата правого
нижнего угла фигуры.\n");
    printf("\t-a/--angle  [<число>] - угол поворота (90/180/270 градусов).\n");

    printf("Четвертый ключ: -c/--circle --Рисование окружности.\n");
    printf("\t-1/--path_1  [<радиус>.<x-координата>.<y-координата>] - радиус и
координата центра окружности.\n");
    printf("\t-2/--path_2  [<x-координата>.<y-координата>.<x-координата>.<y-
координата>] - координаты левого верхнего и правого нижнего углов
фигуры.\n");
    printf("\t-w/--width  [<число>] - толщина линии\n");
    printf("\t-F/--line_color  [<число>.<число>.<число>] - цвет линии (RGB).\n");
    printf("\t-H/--shape  [<число>] - выбор наличия заливки фигуры (0 - не залит, 1
- залит).\n");
    printf("\t-S/--shape_color [<число>.<число>.<число>] - цвет заливки
(RGB).\n");

    printf("-h/--help - вывод справки о работе программы\n");
    printf("-i/--info - вывод подробной информации о bmp-файле\n");
    printf("-o/--output [путь] - файл для вывода (по умолчанию исходный файл,
путь - относительный)\n");
}

void print_bmp_info(BMP_file * bmp_file) {
    printf("Информация о bmp-файле:\n");
    printf("ID = \t\t\t%c%c\n",bmp_file->bhdr.ID[0], bmp_file->bhdr.ID[1]);
    printf("file_size = \t\t%d\n",bmp_file->bhdr.file_size);
    printf("pixel_offset = \t\t%d\n",bmp_file->bhdr.pixel_offset);
```

```
    printf("header_size = \t\t%d\n",bmp_file->dhdr.header_size);
    printf("width = \t\t%d\n",bmp_file->dhdr.width);
    printf("height = \t\t%d\n",bmp_file->dhdr.height);
    printf("planes = \t\t%d\n",bmp_file->dhdr.planes);
    printf("bits_per_pixel = \t%d\n",bmp_file->dhdr.bits_per_pixel);
    printf("compression = \t\t%d\n",bmp_file->dhdr.compression);
    printf("data_size = \t\t%d\n",bmp_file->dhdr.data_size);
    printf("x_pixels_per_meter = \t\t%d\n",bmp_file->dhdr.x_pixels_per_meter);
    printf("y_pixels_per_meter = \t\t%d\n",bmp_file->dhdr.y_pixels_per_meter);
    printf("colors_in_color_table = \t\t%d\n",bmp_file->dhdr.colors_in_color_table);
    printf("important_color_count = \t%d\n",bmp_file->dhdr.important_color_count);
}
```

Файл: *print.h*

```
#pragma once
#include "bmp_processing.h"

void err_limit_img();
void err_value();
void err_color();
int check_color(int color);
void err_find_file();
void count_bits();
void err_version();
void err_file_extension();
void print_bmp_pixels(BMP_file *bmp_file);
void printHelp();
void print_bmp_info(BMP_file * bmp_file);
```

Makefile:

```
all: main.o bmp_processing.o print.o cli.o cw_functions.o
	gcc main.o bmp_processing.o print.o cli.o cw_functions.o -lm -o main
main.o: main.c
	gcc -c -std=c99 main.c
bmp_processing.o: bmp_processing.c
	gcc -c -std=c99 bmp_processing.c
print.o: print.c
	gcc -c -std=c99 print.c
cli.o: cli.c
	gcc -c -std=c99 cli.c
cw_functions.o: cw_functions.c
		gcc -c -std=c99 cw_functions.c
```