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

ОТЧЕТ
ПО КУРСОВОЙ РАБОТЕ

по дисциплине «Программирование»

Тема: Обработка изображений в формате BMP на языке Си с использованием CLI

Студентка гр. 0382 _____ Деткова А.С.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2021

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студентка Деткова А.С.

Группа 0382

Тема работы: обработка изображений в формате BMP на языке Си с использованием CLI

Исходные данные:
Программа принимает на вход файл в формате BMP, список ключей и аргументов к ним. Программа изменяет картинку в соответствии с необходимыми условиями и и возвращает картинку в файл типа bmp. Поддержка осуществляется через CLI (Command Line Interface — терминальный интерфейс).
Предполагаемый объем пояснительной записки:
Не менее 15 страниц.

Дата выдачи задания: 18.04.2021

Дата сдачи реферата: 23.05.2021

Дата защиты реферата: 25.05.2021

Студентка _____ Деткова А.С.

Преподаватель _____ Берленко Т.А.

АННОТАЦИЯ

В ходе курсовой работы была написана программа, которая получает на вход файл типа .bmp и которая реализует требуемый функционал. Поддержка осуществляется через CLI (Command Line Interface — терминальный интерфейс) с помощью библиотеки getopt.h. Программа представляет следующие опции: вывод информации о работе программы (меню помощи); вывод информации о bmp файле; рисование прямоугольника с заданными пользователем толщиной рамки и ее цветом, а также заливкой; рисование окружности залитой или нет с с заданными пользователем толщиной рамки и ее цветом; поворот изображения (части) на 90/180/270 градусов; рисование рамки (одной из 4 представленных на выбор).

SUMMARY

In the course of the course work, a program was written that receives a .bmp file as input and which implements the required functionality, implemented through the CLI (command line interface - terminal interface) using the getopt.h library. The program presents the following options :. programs (help menu); output information about the BMP file; drawing a rectangle frame with a given thickness and color, as well as; drawing a circle filled or not with a given thickness and color; rotation of the image (part) by 90/180/270 degrees; drawing a frame (one of the 4 presented to choose from).

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Кратко описать цель работы, основные задачи им методы их решения.

Необходимо написать программу, которая обрабатывает файл-изображение в формате bmp, и обрабатывает его в соответствии с запросом пользователя.

Программа стабильно работает на ОС Linux (Ubuntu). Была написана с помощью IDE Clion, протестирована с помощью командной строки терминала.

Программа предоставляет CLI — терминальный интерфейс.

Отлавливаются все случаи некорректного обращения с программой и выводятся соответствующие ошибки и предупреждения.

1. Цель и задачи работы

Цель: изучить особенности работы с bmp файлами и написать программу, решающую поставленные задачи.

Задачи:

- обеспечить бинарное считывание файла с изображением
- реализовать структуры данных для хранения информационных полей файла, а также массива пикселей
- написать функции по обработке структуры
- сохранить обработанные структуры в новый файл и вернуть обработанный бинарный файл с изображением пользователю
- реализовать CLI

2. Задание курсовой работы

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

ВЫПОЛНЕНИЕ РАБОТЫ

1. Ход решения

Используются следующие заголовочные файлы: *stdio.h* — описывает функции ввода/вывода; *stdlib.h* - для работы с функциями, работающими с динамической памятью; *stdint.h* — описания целочисленных типов; *getopt.h* — работа с CLI; *string.h* — обработка строк.

Типы данных: *struct BitmapFileHeader* — структура, поля которой являются полями бинарного файла типа *bmp* — файлового заголовка; struct *BitmapInfoHeader* — структура, поля которой — поля бинарного файла типа *bmp* — информационного заголовка; *struct RGB* — структура, поля которой являются компонентами *R*, *G* и *B* каждого пикселя, массив таких структур хранит информацию о пикселях изображений; *struct image* — объединяет все структуры в одну (для удобства). При считывании в структру *RGB* применяется выравнивание, чтобы ширина строки была кратна 4 байтам: (4 — ([ширина изображения]*3)%4)%4.

В функции *main()* происходит обработка полученных ключей и значений, вызов требуемых функций. Используется *getopt()*.

2. Функции

1. void drawPoint(RGB *elem, unsigned char r, unsigned char g, unsigned char b) — записывает информацию о пикселе в соответствующий элемент структуры. RGB *elem — указатель на элемент массива пикселей, r — значение для red-компоненты, g — значение для green-компоненты, b — значение для blue-компоненты.
2. void printFileHeader(BitmapFileHeader header) — вывод информации заголовка. BitmapFileHeader header — заголовок файла с изображением.

3. void printInfoHeader(BitmapInfoHeader header) — вывод информации информационного заголовка файла. BitmapInfoHeader header — информационный заголовок файла.
4. int readFile(const char * str, image *img) — считывание файла в структуру файла image с использованием функций fread(), fclose() и fopen() с типом операции «rb» - бинарное чтение из файла. Динамически выделяет память под массив пикселей. const char * str — имя считываемого файла, image *img — указатель на структуру bmp-файла. Возвращает 0, если удалось считать файл, и -1, если не удалось.
5. void writeFile(const char * str, image *img) — запись информации о файле в бинарный файл с помощью функций fwrite(), fclose() и fopen() с типом операции «wb» - бинарная запись в файл. const char * str — имя файла, под которым будет храниться изображение, image *img — указатель на структуру bmp-файла.
6. void drawRectangle(RGB **arr, unsigned int W, unsigned int H, int x1, int y1, int x2, int y2, int w, unsigned char rl, unsigned char gl, unsigned char bl) — рисует незалитый прямоугольник с заданной шириной и цветом линии. RGB **arr — указатель на массив пикселей, unsigned int W, unsigned int H — ширина и высота изображения, int x1, int y1, int x2, int y2 — координаты левого верхнего и правого нижнего угла прямоугольника, int w — ширина линии, unsigned char rl, unsigned char gl, unsigned char bl — значение каждой составляющей из RGB. У подходящего пикселя из массива пикселей меняется значение полей структуры составляющих RGB.
7. void drawFilledRectangle(RGB **arr, unsigned int W, unsigned int H, int x1, int y1, int x2,int y2, int w, unsigned char rl, unsigned char gl, unsigned char bl, unsigned char rf, unsigned char gf, unsigned char bf) — рисует залитый прямоугольник с заданной шириной и цветом линии и цветом заливки. RGB **arr — указатель на массив пикселей, unsigned int W, unsigned int H — ширина и высота изображения, int x1, int y1, int x2, int y2 —

координаты левого верхнего и правого нижнего угла прямоугольника, int w — ширина линии, unsigned char rl, unsigned char gl, unsigned char bl — значение каждой составляющей из RGB для линии, unsigned char rf, unsigned char gf, unsigned char bf - значение каждой составляющей из RGB для заливки. У подходящего пикселя из массива пикселей меняется значение полей структуры составляющих RGB.

8. void drawLine(RGB **arr, unsigned int H, int x0, int y0, int x1, int y1, int w, unsigned char r, unsigned char g, unsigned char b) — рисует линию с использованием алгоритма Брезенхема. RGB **arr — указатель на массив пикселей, unsigned int W, unsigned int H — ширина и высота изображения, int x1, int y1, int x2, int y2 — координаты начала и конца, int w — ширина линии, unsigned char r, unsigned char g, unsigned char b — значение каждой составляющей из RGB для линии.
9. void drawRing(RGB **arr, unsigned int H, unsigned int W, int R, int x0, int y0, int w, unsigned char r, unsigned char g, unsigned char b) — рисует кольцо заданной толщины и цвета без заливки. RGB **arr — указатель на массив пикселей, unsigned int W, unsigned int H — ширина и высота изображения, int R, int x0, int y0 — радиус, абсцисса и ордината центра , int w — ширина линии, unsigned char r, unsigned char g, unsigned char b — значение каждой составляющей из RGB для линии. Использует алгоритм Брезенхема.
10. void drawCircle(RGB **arr, unsigned int H, unsigned int W, int R, int x0, int y0, int w, unsigned char rl, unsigned char gl,unsigned char bl, unsigned char rf, unsigned char gf, unsigned char bf) - рисует круг с заливкой заданной толщины и цвета линии и заливки. RGB **arr — указатель на массив пикселей, unsigned int W, unsigned int H — ширина и высота изображения, int R, int x0, int y0 — радиус, абсцисса и ордината центра , int w — ширина линии, unsigned char rl, unsigned char gl,unsigned char bl unsigned char rl, unsigned char gl,unsigned char bl — значение каждой составляющей из RGB для линии, unsigned char rf, unsigned char gf,

unsigned char bf — значение каждой составляющей из RGB для заливки. Использует алгоритм Брезенхема.

11. void drawFrame1(RGB **arr, int H, int W, int x0, int y0, int x1, int y1, int w, unsigned char r, unsigned char g, unsigned char b) — рисует двойную рамку. RGB **arr — указатель на массив пикселей, int W, int H — ширина и высота изображения, int x0, int y0, int x1, int y1 — значения крацних координат изображения (1,1,ширина,высота), int w — ширина линии, unsigned char r, unsigned char g,unsigned char b — значение каждой составляющей из RGB для рамки.
12. void drawFrame2(RGB **arr, int H, int W, int w, unsigned char r, unsigned char g, unsigned char b) — рисует рамку заданного цвета и толщины из точек. RGB **arr — указатель на массив пикселей, int W, int H — ширина и высота изображения, int w — ширина линии, unsigned char r, unsigned char g,unsigned char b — значение каждой составляющей из RGB для рамки.
13. void drawFrame3(RGB **arr, int H, int W, int w, unsigned char r, unsigned char g, unsigned char b) — рисует рамку заданного цвета и ширины из полосочек. RGB **arr — указатель на массив пикселей, int W, int H — ширина и высота изображения, int w — ширина линии, unsigned char r, unsigned char g,unsigned char b — значение каждой составляющей из RGB для рамки.
14. void drawFrame4(RGB **arr, int H, int W, int w, unsigned char r, unsigned char g, unsigned char b) — рисует рамку заданного цвета и ширины из кружочков. RGB **arr — указатель на массив пикселей, int W, int H — ширина и высота изображения, int w — ширина линии, unsigned char r, unsigned char g,unsigned char b — значение каждой составляющей из RGB для рамки.
15. void turnImage(image *img, int grade, int x0, int y0, int x1, int y1) — поворачивает изображение или его часть на заданный угол. image *img — указатель на структуру изображения со всеми информационными

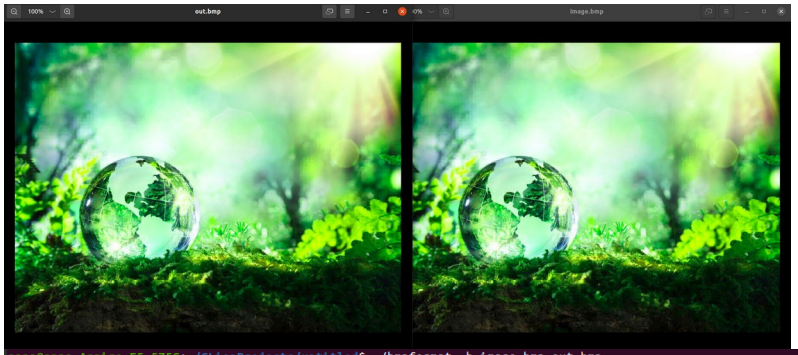
полями и массивом символов, int grade — угол поворота , int x0, int y0, int x1, int y1 — координаты левого верхнего и правого нижнего углов прямоугольной области для поворота.

16. void printHelp() - выводит информацию о возможных действиях в программе.
17. int main(int argc, char *argv[]) - разбирает полученные на вход ключи, а также данные о файле (для считывания и сохранения изображения), вызывает функции, требуемые в зависимости от ситуации.

ТЕСТИРОВАНИЕ

Вывод информации о вводе данных

```
anna@anna-Aspire-E5-575G:~/CLionProjects/untitled$ ./bmpformat -h
bmpformat - program that supports CLI and works with bmp format files;
Only bmp files version 3 are supported (40 bytes BITMAPINFOHEADER);
bmp files with color table AREN'T supported;
Image depth = 24 bites per pixel, without compression; number of planes = 1;
        Format of input: ./bmpformat [instruction] [name of input file] -[key1]/--[long_ke1] [arg1] ... [name of outpu
t file] (the last argument)

        Instructions:

        --DrawRectangle/-R [name of file] - drawing filled or unfilled rectangle:
                -l/--leftcorner [x0.y0]- left corner coordinates of rectangle.
                -r/--rightcorner [x1.y1]- right corner coordinates of rectangle.
                -w/--width [number] - width of rectangle's frame (==0 by default).
                -c/--color1 [255.255.255] - color of frame if format RGB (== 0.0.0 by default).
                -f/--filling [0/1] - fill, if there is, then 1, else - 0 (==0 by default).
                -C/--color2 [255.255.255] - color of filling in format RGB (== 0.0.0 by default).
        --Frame/-F [name of file] - drawing of frame:
                -s/--style [1/2/3/4] - choose the style of frame:
                        1 - double frame; 2 - frame in points; 3 - stripes; 4 - rings;
                -c/--color [255.255.255] - color of frame in format RGB (== 0.0.0 by default).
                -w/--width [number] - width of frame (== 0 by default).
        --TurnImage/-T [name of file] - rotate the image by the specified angle:
                -l/--leftcorner [x0.y0]- left corner coordinates of zone (== 1.1 by default).
                -r/--rightcorner [x1.y1]- right corner coordinates of zone (== width.height by default).
                -a/--angle [0/90/180/270] - angle (== 0 by default).
        --DrawCircle/-L [name of file] - draw a filled or unfilled circle:
                -p/--coordinates [x0.y0.R/x0.y0.x1.y1] - coordinates of circle: coordinates of centre and radius or
                        the upper left and lower right corners of the square where the circle id inscribed.
                -w/--width [number] width of the ring (== 0 by default).
                -c/--color1 [255.255.255] - color of ring in format RGB (== 0.0.0 by default).
                -f/--filling [0/1] - fill the circle - 1-filled circle; 0-unfilled (== 0 by default).
                -C/--color2 [255.255.255] - color of filling in format RGB (== 0.0.0 by default).
        --info/-i [name of file] - information about file.
        -h/--help/-? - about program.
```



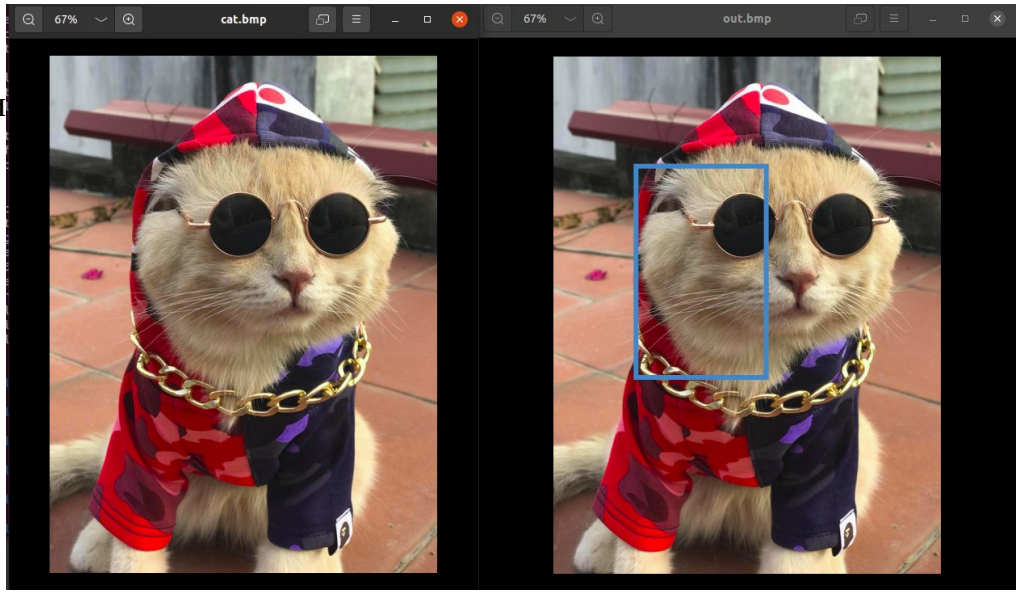

```
anna@anna-Aspire-E5-575G:~/CLionProjects/untitled$ ./bmpformat -h image.bmp out.bmp
bmpformat - program that supports CLI and works with bmp format files;
Only bmp files version 3 are supported (40 bytes BITMAPINFOHEADER);
bmp files with color table AREN'T supported;
Image depth = 24 bites per pixel, without compression; number of planes = 1;
        Format of input: ./bmpformat [instruction] [name of input file] -[key1]/--[long_ke1] [arg1] ... [name of outpu
t file] (the last argument)

        Instructions:

        --DrawRectangle/-R [name of file] - drawing filled or unfilled rectangle:
                -l/--leftcorner [x0.y0]- left corner coordinates of rectangle.
                -r/--rightcorner [x1.y1]- right corner coordinates of rectangle.
                -w/--width [number] - width of rectangle's frame (==0 by default).
                -c/--color1 [255.255.255] - color of frame if format RGB (== 0.0.0 by default).
                -f/--filling [0/1] - fill, if there is, then 1, else - 0 (==0 by default).
                -C/--color2 [255.255.255] - color of filling in format RGB (== 0.0.0 by default).
        --Frame/-F [name of file] - drawing of frame:
                -s/--style [1/2/3/4] - choose the style of frame:
                        1 - double frame; 2 - frame in points; 3 - stripes; 4 - rings;
                -c/--color [255.255.255] - color of frame in format RGB (== 0.0.0 by default).
                -w/--width [number] - width of frame (== 0 by default).
        --TurnImage/-T [name of file] - rotate the image by the specified angle:
                -l/--leftcorner [x0.y0]- left corner coordinates of zone (== 1.1 by default).
                -r/--rightcorner [x1.y1]- right corner coordinates of zone (== width.height by default).
                -a/--angle [0/90/180/270] - angle (== 0 by default).
        --DrawCircle/-L [name of file] - draw a filled or unfilled circle:
                -p/--coordinates [x0.y0.R/x0.y0.x1.y1] - coordinates of circle: coordinates of centre and radius or
                        the upper left and lower right corners of the square where the circle id inscribed.
                -w/--width [number] width of the ring (== 0 by default).
                -c/--color1 [255.255.255] - color of ring in format RGB (== 0.0.0 by default).
                -f/--filling [0/1] - fill the circle - 1-filled circle; 0-unfilled (== 0 by default).
                -C/--color2 [255.255.255] - color of filling in format RGB (== 0.0.0 by default).
        --info/-i [name of file] - information about file.
        -h/--help/-? - about program.
```

Вывод информации о вводе и ввод/вывод файла

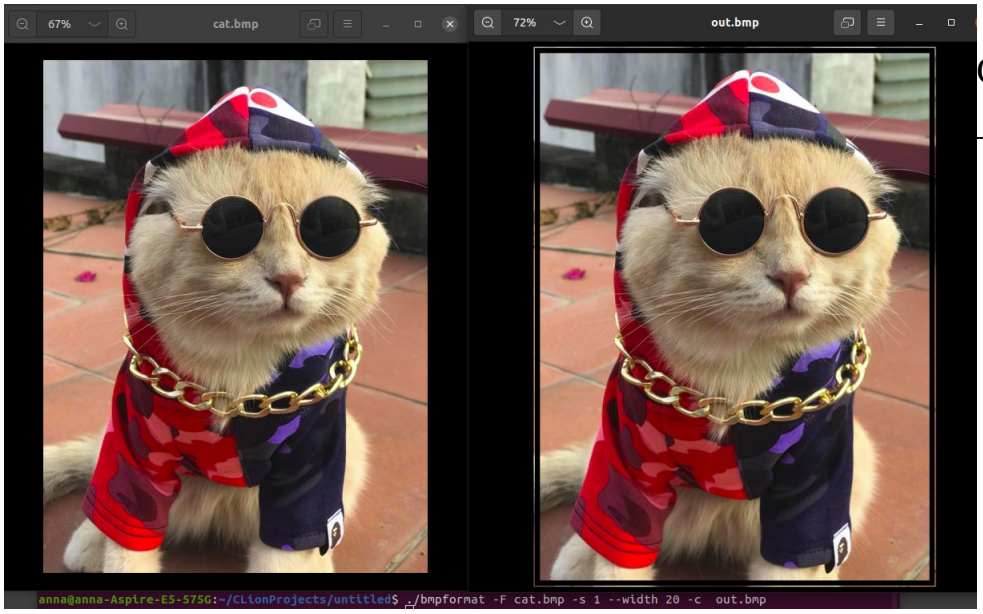
```
anna@anna-Aspire-E5-575G:~/CLionProjects/untitled$ ./bmpformat -h ige.bmp out.bmp
Error: file not found
Try again!
bmpformat - program that supports CLI and works with bmp format files;
Only bmp files version 3 are supported (40 bytes BITMAPINFOHEADER);
bmp files with color table AREN'T supported;
Image depth = 24 bites per pixel, without compression; number of planes = 1;
        Format of input: ./bmpformat [instruction] [name of input file] -[key1]/--[long_ke1] [arg1] ... [name of outpu
t file] (the last argument)

        Instructions:

        --DrawRectangle/-R [name of file] - drawing filled or unfilled rectangle:
                -l/--leftcorner [x0.y0]- left corner coordinates of rectangle.
                -r/--rightcorner [x1.y1]- right corner coordinates of rectangle.
                -w/--width [number] - width of rectangle's frame (==0 by default).
                -c/--color1 [255.255.255] - color of frame if format RGB (== 0.0.0 by default).
                -f/--filling [0/1] - fill, if there is, then 1, else - 0 (==0 by default).
                -C/--color2 [255.255.255] - color of filling in format RGB (== 0.0.0 by default).
        --Frame/-F [name of file] - drawing of frame:
                -s/--style [1/2/3/4] - choose the style of frame:
                        1 - double frame; 2 - frame in points; 3 - stripes; 4 - rings;
                -c/--color [255.255.255] - color of frame in format RGB (== 0.0.0 by default).
                -w/--width [number] - width of frame (== 0 by default).
        --TurnImage/-T [name of file] - rotate the image by the specified angle:
                -l/--leftcorner [x0.y0]- left corner coordinates of zone (== 1.1 by default).
                -r/--rightcorner [x1.y1]- right corner coordinates of zone (== width.height by default).
                -a/--angle [0/90/180/270] - angle (== 0 by default).
        --DrawCircle/-L [name of file] - draw a filled or unfilled circle:
                -p/--coordinates [x0.y0.R/x0.y0.x1.y1] - coordinates of circle: coordinates of centre and radius or
                        the upper left and lower right corners of the square where the circle id inscribed.
                -w/--width [number] width of the ring (== 0 by default).
                -c/--color1 [255.255.255] - color of ring in format RGB (== 0.0.0 by default).
                -f/-filling [0/1] - fill the circle - 1-filled circle; 0-unfilled (== 0 by default).
                -C/--color2 [255.255.255] - color of filling in format RGB (== 0.0.0 by default).
        --info/-i [name of file] - information about file.
        -h/--help/-? - about program.
```

Файл не найден, вывод информации о вводе

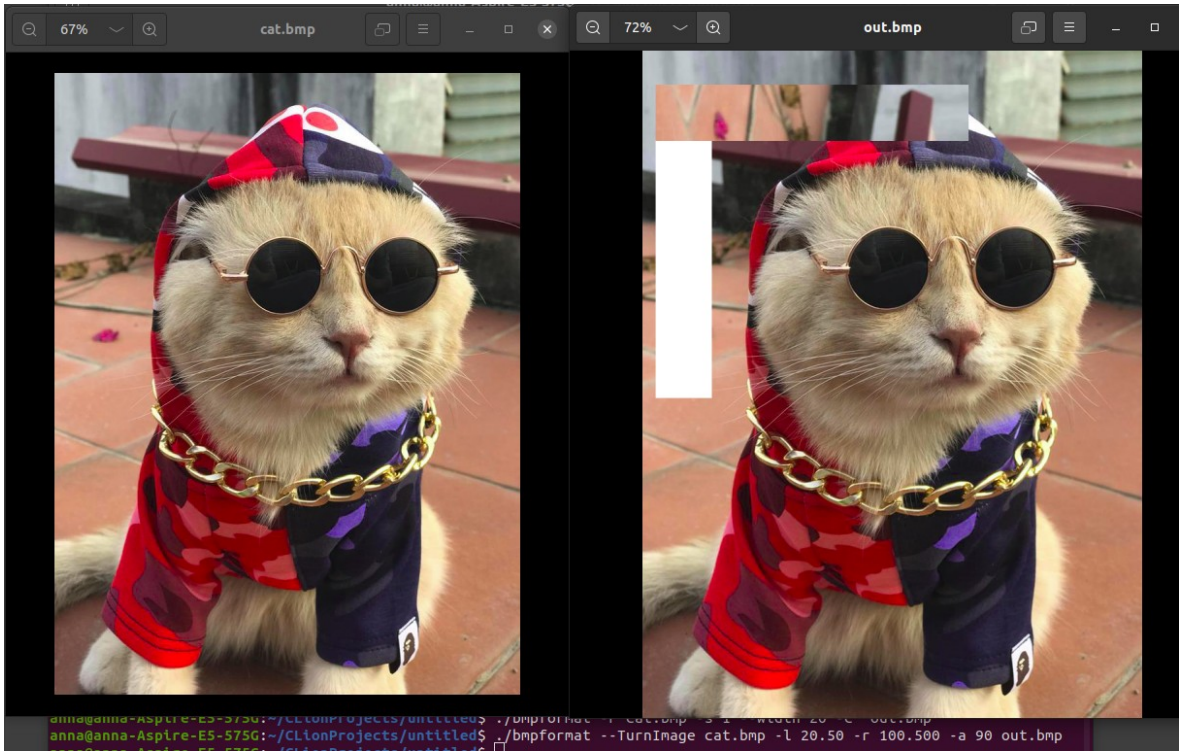
Ввод файла и рисование незакрашенного прямоугольника




```
anna@anna-Aspire-E5-575G:~/CLionProjects/untitled$ ./bmpformat --DrawRectangle cat.bmp -l 150.200 --rightcorner 400.60
0 -w 9 --color1 50.150.200 out.bmp
```

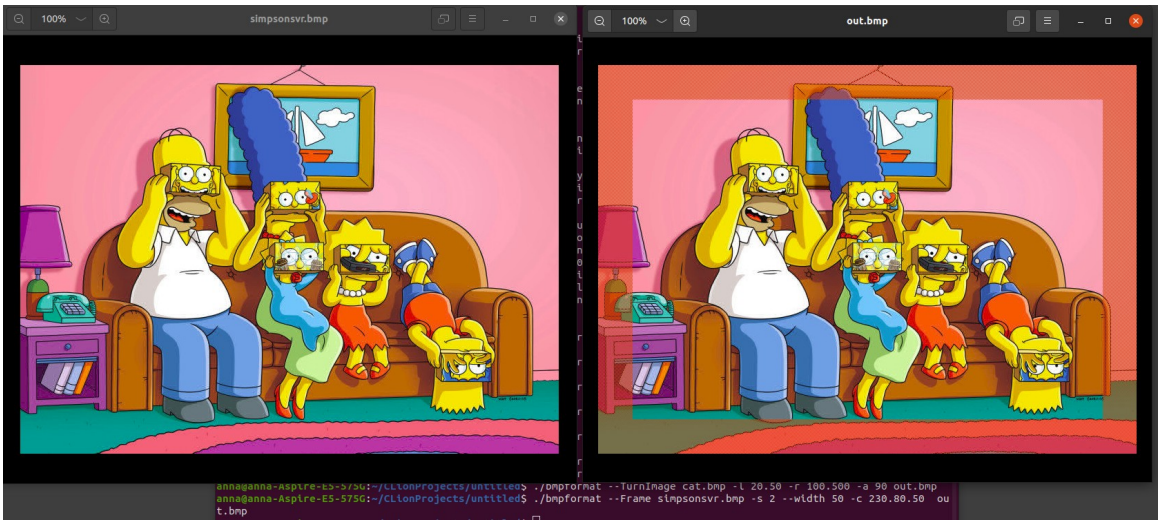


Создание рамки в стиле 1 — двойная рамка

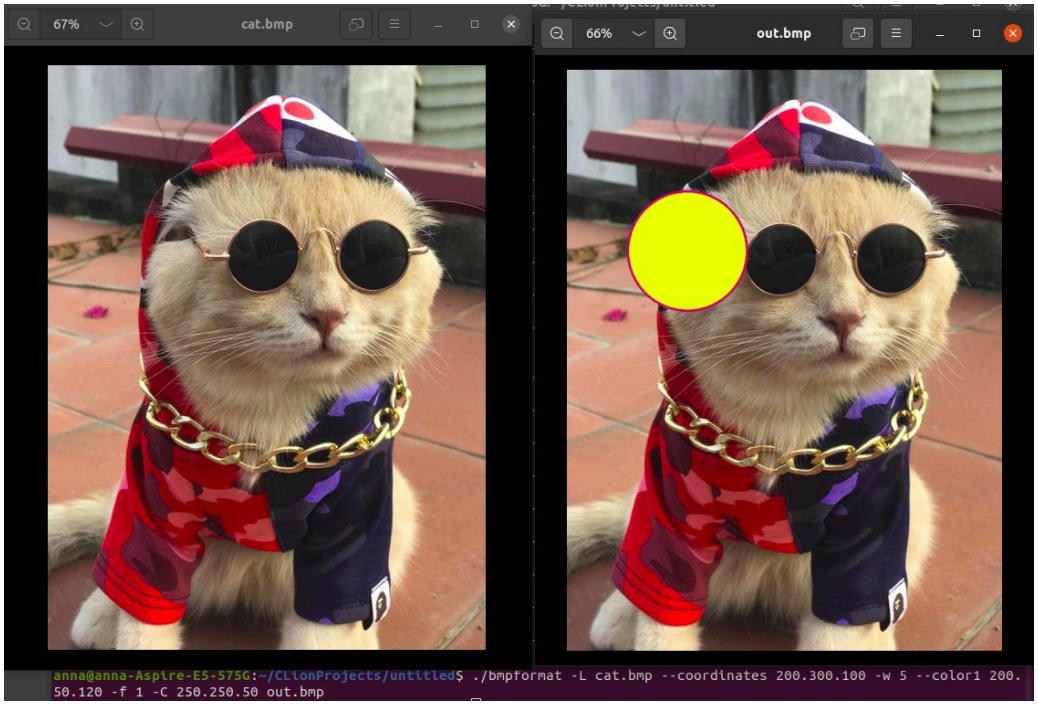


Поворот части изображения на 90 градусов

Рисование рамки в стиле 2 - точки



Поворот всего изображения на 180 градусов



Рисование закрашенной окружности

ЗАКЛЮЧЕНИЕ

Были изучены способы работы с bmp файлами.

По итогу была разработана программа, получающая имя файла от пользователя, выполняющая необходимую работу и возвращающая измененное изображение в бинарный файл. Поддержка осуществлена с помощью CLI.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. https://docs.microsoft.com/ru-ru/cpp/cpp/main-function-command-line-args?view=msvc-160
2. http://www.c-cpp.ru/
3. https://ru.wikipedia.org/wiki/BMP
4. https://jenyay.net/Programming/Bmp
5. https://api-2d3d-cad.com/bmp/
6. https://opensource.com/article/19/5/how-write-good-c-main-function

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название в репозитории: *main.c*

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <getopt.h>
#include <string.h>

#pragma pack (push, 1)
typedef struct
{
    uint16_t signature;
    uint32_t fileSize;
    uint16_t reserved1;
    uint16_t reserved2;
    uint32_t filePxlArrOffset;
} BitmapFileHeader;

typedef struct
{
    uint32_t headerSize; //size of header
    uint32_t width; //width of picture
    uint32_t height; //height of picture
    uint16_t planes; //слои
    uint16_t bitsPerPixel; //бит на пиксель
    uint32_t compression; //сжатие
       uint32_t  imageSize;  //размер  изображения(непосредственно  самой
картинки без заголовков)
    uint32_t xPixelsPerMeter;
    uint32_t yPixelsPerMeter;
    uint32_t colorsInColorTable; //количество цветов в палитре
    uint32_t importantColorCount; //количество важных цветов в палитре
} BitmapInfoHeader;

typedef struct
{
    uint8_t B;
    uint8_t G;
    uint8_t R;
} RGB;

typedef struct
{
    BitmapFileHeader bmfh;
    BitmapInfoHeader bmih;
    RGB ** pxlsArr;
} image;
#pragma pack (pop)

void drawPoint(RGB *elem, unsigned char r, unsigned char g, unsigned char
b){
    elem->B = b;
    elem->G = g;
    elem->R = r;
```

```
}

void printFileHeader(BitmapFileHeader header){
    printf("signature:\t%x (%hu)\n", header.signature, header.signature);
    printf("filesize:\t%x (%u)\n", header.fileSize, header.fileSize);
    printf("reserved1:\t%x (%hu)\n", header.reserved1, header.reserved1);
    printf("reserved2:\t%x (%hu)\n", header.reserved2, header.reserved2);
         printf("pixelArrOffset:\t%x  (%u)\n",  header.filePxlArrOffset,
header.filePxlArrOffset);
}

void printInfoHeader(BitmapInfoHeader header){
                printf("headerSize:\t%x    (%u)\n",    header.headerSize,
header.headerSize);
    printf("width:     \t%x (%u)\n", header.width, header.width);
    printf("height:    \t%x (%u)\n", header.height, header.height);
    printf("planes:    \t%x (%hu)\n", header.planes, header.planes);
             printf("bitsPerPixel:\t%x   (%hu)\n",   header.bitsPerPixel,
header.bitsPerPixel);
               printf("compression:\t%x    (%u)\n",    header.compression,
header.compression);
    printf("imageSize:\t%x (%u)\n", header.imageSize, header.imageSize);
         printf("xPixelsPerMeter:\t%x  (%u)\n",  header.xPixelsPerMeter,
header.xPixelsPerMeter);
         printf("yPixelsPerMeter:\t%x  (%u)\n",  header.yPixelsPerMeter,
header.yPixelsPerMeter);
     printf("colorsInColorTable:\t%x (%u)\n", header.colorsInColorTable,
header.colorsInColorTable);
    printf("importantColorCount:\t%x (%u)\n", header.importantColorCount,
header.importantColorCount);
}

int readFile(const char * str, image *img){
    FILE *f = fopen(str,"rb");
    if(!f){
        printf("Error: file not found\n");
        return 1;
    }
    fread(&(img->bmfh),1,sizeof(BitmapFileHeader),f);
    if(img->bmfh.signature != 0x4d42){
        printf("Error: unsupported format\n");
        return 1;
    }
    fread(&(img->bmih),1,sizeof(BitmapInfoHeader),f);
    if(img->bmih.headerSize!=40){
        printf("Error: Unsupported file format\n");
        return 1;
    }
    if(img->bmih.planes!=1){
        printf("Error: Unsupported quantity of planes\n");
        return 1;
    }
    if(img->bmih.bitsPerPixel!=24){
        printf("Error: Unsupported bit rate\n");
        return 1;
    }
    if(img->bmih.compression){
        printf("Error: Unsupported compression\n");
```

```
        return 1;
    }
    if(img->bmih.colorsInColorTable || img->bmih.importantColorCount){
        printf("Error: Images with color table are unsupported\n");
        return 1;
    }
    img->pxlsArr = malloc((img->bmih.height)*sizeof(RGB *));
    if(!(img->pxlsArr)){
        printf("Error: memory isn't available");
        return 1;
    }
    for(int i=0; i<img->bmih.height; i++){
            img->pxlsArr[i] = malloc((img->bmih.width)*sizeof(RGB) + (4-
((img->bmih.width)*3)%4)%4);
        if(!(img->pxlsArr[i])){
            printf("Error: memory isn't available");
            return 1;
        }
        fread(img->pxlsArr[i],1,(img->bmih.width)*sizeof(RGB) + (4-((img-
>bmih.width)*3)%4)%4,f);
    }
    fclose(f);
    return 0;
}

void writeFile(const char * str, image *img){
    FILE * ff = fopen(str, "wb");

    fwrite(&(img->bmfh), 1, sizeof(BitmapFileHeader), ff);
    fwrite(&(img->bmih), 1, sizeof(BitmapInfoHeader), ff);
       unsigned int w = ((img->bmih).width)*sizeof(RGB) + (4-((img-
>bmih).width*3)%4)%4;
    for(int i=0; i<(img->bmih).height; i++){
        fwrite((img->pxlsArr)[i],1,w,ff);
    }
    fclose(ff);
    for(int i=0; i<(img->bmih).height; i++){
        free(img->pxlsArr[i]);
    }
    free(img->pxlsArr);
}

void drawRectangle(RGB **arr, unsigned int W, unsigned int H, int x1, int
y1, int x2,
                       int y2, int w, unsigned char rl, unsigned char gl,
unsigned char bl){
    //x1,y1 - координаты левого верхнего угла; x2,y2 - координаты правого
нижнего угла
    //w - ширина линии прямоугольника; W, H - ширина и высота картинки;
    if(x1>x2 || y1>y2){
        printf("Error: Put incorrect meaning of coordinates");
        return;
    }
     if(x1>W || x2>W || x1<0 || x2<0 || y1>H || y2>H || y1<0 || y2<0 ||
x1>x2 || y1>y2){
        printf("Error: Incorrect meaning of rectangle coordinates\n");
        return;
    }
```

```
    if(w<0 || w>=(abs(x1-x2)) || w>=(abs(y1-y2))){
        printf("Error: Incorrect width\n");
        return;
    }

    for(unsigned int i=(H-y2); i<=(H-y1); i++){
        for(int j=0; j<w; j++){
            drawPoint(&(arr[i][x2-j-1]), rl, gl, bl);
            drawPoint(&(arr[i][x1+j-1]), rl, gl, bl);
        }
    }
    for(unsigned int i=x1; i<=x2; i++){
        for(int j=0; j<w; j++){
            drawPoint(&(arr[H-y2+j][i-1]), rl, gl, bl);
            drawPoint(&(arr[H-y1-j][i-1]), rl, gl, bl);
        }
    }

}

void drawFilledRectangle(RGB **arr, unsigned int W, unsigned int H, int
x1, int y1, int x2,
                         int y2, int w, unsigned char rl, unsigned char
gl, unsigned char bl,
                           unsigned char rf, unsigned char gf, unsigned
char bf) {
    if(x1>x2 || y1>y2){
        printf("Error: Put incorrect meaning of coordinates");
        return;
    }
     if(x1>W || x2>W || x1<0 || x2<0 || y1>H || y2>H || y1<0 || y2<0 ||
x1>x2 || y1>y2){
        printf("Error: Incorrect meaning of rectangle coordinates\n");
        return;
    }
    if(w<0 || w>=(abs(x1-x2)) || w>=(abs(y1-y2))){
        printf("Error: Incorrect width\n");
        return;
    }

    drawRectangle(arr,W,H,x1,y1,x2,y2,w,rl,gl,bl);

    for(unsigned int i=H-y2+w; i<=(H-y1-w); i++)
        for(int j=x1+w; j<=(x2-w); j++){
            drawPoint(&(arr[i][j-1]), rf, gf, bf);
        }
}

void drawLine(RGB **arr, unsigned int H, int x0, int y0, int x1, int y1,
int w,
              unsigned char r, unsigned char g, unsigned char b){

    int deltaX = x1 - x0;
    int deltaY = y1 - y0;
    int absDeltaX = abs(deltaX);
    int absDeltaY = abs(deltaY);
    int inc = 0;
    int alfaX = 0, alfaY = 0;
```

```
    if(x1>x0){
        alfaX = 1;
    }else if(x1<x0){
        alfaX = -1;
    }
    if(y1>y0){
        alfaY = 1;
    }else if(y1<y0){
        alfaY = -1;
    }

    if (absDeltaX >= absDeltaY) {
        int x = x0;
        int y = y0;
        for (int i = 0; i <= absDeltaX; i++) {
            for(int j=0; j<w; j++) {
                drawPoint(&(arr[H - y - j][x - 1]), r, g, b);
            }
            x = x + alfaX;
            inc = inc + absDeltaY;
            if (inc >= absDeltaX) {
                inc = inc - absDeltaX;
                y = y + alfaY;
            }
        }
    } else {
        int x = x0;
        int y = y0;
        for (int i=0; i <= absDeltaY; i++) {
            for(int j=0; j<w; j++){
                drawPoint(&(arr[H-y][x-1+j]), r, g, b);
            }
            y = y + alfaY;
            inc = inc + absDeltaX ;
            if (inc >= absDeltaY) {
                inc = inc - absDeltaY;
                x = x + alfaX;
            }
        }
    }
}

void drawRing(RGB **arr, unsigned int H, unsigned int W, int R, int x0,
int y0, int w, unsigned char r, unsigned char g,
              unsigned char b){

     if(R<=0 || x0<=0 || y0<=0 || x0>W || y0>H || ((x0-R)<=0) || ((y0-
R)<=0) ||
    ((R+x0)>W) || ((R+y0)>H) || (w<0) || (w>=(2*R))){
        printf("Error: Invalid parameters.\n");
        return;
    }

    int x = R;
    int y = 0;
    int dd = x*x + y*y;
    int rr = R*R;
```

```
    while(x>=y){
        for(int j=0; j<w; j++){
            drawPoint(&(arr[(int)H - (y0-y)][x0+x-j-1]),r,g,b);//1 8
            drawPoint(&(arr[(int)H - (y0-x+j)][y+x0-1]),r,g,b);//2 8
            drawPoint(&(arr[(int)H - (y0+y)][x+x0-j-1]),r,g,b);//-1 8
            drawPoint(&(arr[(int)H - (y0+x-j)][y+x0-1]),r,g,b);//-2 8
            drawPoint(&(arr[(int)H - (y0-y)][x0-x+j-1]),r,g,b);//4 8
            drawPoint(&(arr[(int)H - (y0-x+j)][x0-y-1]),r,g,b);//3 8
            drawPoint(&(arr[(int)H - (y0+y)][x0-x+j-1]),r,g,b);//-4 8
            drawPoint(&(arr[(int)H - (y0+x-j)][x0-y-1]),r,g,b);//-3 8
        }
        dd += 2*y + 1;
        if(abs(rr-dd)>abs(rr-(dd-2*x-1)) && dd>rr){
            y += 1;
            dd -= 2*x - 1;
            x -= 1;
        } else {
            y += 1;
        }
    }
}

void drawCircle(RGB **arr, unsigned int H, unsigned int W, int R, int x0,
int y0, int w, unsigned char rl, unsigned char gl,
                  unsigned char bl, unsigned char rf, unsigned char gf,
unsigned char bf){
     if(R<=0 || x0<=0 || y0<=0 || x0>=W || y0>=H || ((x0-R)<=0) || ((y0-
R)<=0) ||
       ((R+x0)>=W) || ((R+y0)>=H) || (w<0) || (w>=(2*R))){
        printf("Error: Invalid parameters.\n");
        return;
    }
    drawRing(arr,H,W,R,x0,y0,w,rl,gl,bl);
    drawRing(arr,H,W,R-w+1,x0,y0,R-w+2,rf,gf,bf);
}

void drawFrame1(RGB **arr, int H, int W, int x0, int y0, int x1, int y1,
int w,
                unsigned char r, unsigned char g, unsigned char b){
    if(w>=W/2 || w>=H/2 || w<0){
        printf("Error: Incorrect width of frame");
        return;
    }
    if(w==0){
        return;
    }
    if(w%2){
        drawRectangle(arr,W,H,x0,y0,x1,y1,w/2,r,g,b);
        drawRectangle(arr,W,H,x0+1+w/2,y0+1+w/2,x1-(1+w/2),y1-(1+w/2),w/2
,r,g,b);
    } else {
        drawRectangle(arr,W,H,x0,y0,x1,y1,w/2-1,r,g,b);
        drawRectangle(arr,W,H,x0+1+w/2,y0+1+w/2,x1-(1+w/2),y1-(1+w/2),w/2
-1,r,g,b);
    }
} //doubleFrame

void drawFrame2(RGB **arr, int H, int W, int w,
```

```
                unsigned char r, unsigned char g, unsigned char b) {
    if(w>=W/2 || w>=H/2 || w<0){
        printf("Error: Incorrect width of frame");
        return;
    }
    if(w==0){
        return;
    }
    for (int i = 1; i <= w; i++) {
        for (int j = 0; j < W; j++) {
            if ((i % 2 && j % 2) || (i % 2 == 0 && j % 2 == 0)) {
                drawPoint(&(arr[H - i][j]), r, g, b);
                drawPoint(&(arr[i - 1][j]), r, g, b);
            }
        }
    }
    for (int i = 1; i <= H; i++) {
        for (int j = 0; j < w; j++) {
            if ((i % 2 && j % 2) || (i % 2 == 0 && j % 2 == 0)) {
                drawPoint(&(arr[H - i][j]), r, g, b);
                drawPoint(&(arr[H - i][W - j]), r, g, b);
            }
        }
    }
}//Frame in points

void drawFrame3(RGB **arr, int H, int W, int w, unsigned char r, unsigned
char g, unsigned char b){
    if(w>=W/2 || w>=H/2 || w<0){
        printf("Error: Incorrect width of frame");
        return;
    }
    if(w==0){
        return;
    }
    for(int i=1; i<=H; i+=8){
        if((i+5)<H){
            drawLine(arr,H,1,i,1,i+3,w,r,g,b);
            drawLine(arr,H,W-w+1,i,W-w+1,i+3,w,r,g,b);
        }
    }
    for(int i=1; i<=W; i+=8){
        if((i+5)<=W){
            drawLine(arr,H,i,1,i+3,1,w,r,g,b);
            drawLine(arr,H,i,H-w+1,i+3,H-w+1,w,r,g,b);
        }
    }
}

void drawFrame4(RGB **arr, int H, int W, int w, unsigned char r, unsigned
char g, unsigned char b){
    if(w>=W/2 || w>=H/2 || w<0){
        printf("Error: Incorrect width of frame");
        return;
    }
    if(w==0){
        return;
    }
```

```
    if(w%2){
        int R = w/2;
        for(int i=1; i<=H; i+=R){
            if((i-R)>0 && (i+R)<=H){
                drawRing(arr,H,W,R,R+1,i,1,r,g,b);
                drawRing(arr,H,W,R,W-R,i,1,r,g,b);
            }
        }
        for(int i=1; i<=W; i+=R){
            if((i-R)>0 && (i+R)<=W){
                drawRing(arr,H,W,R,i,R+1,1,r,g,b);
                drawRing(arr,H,W,R,i,H-R,1,r,g,b);
            }
        }
    } else {
        int R = w/2-1;
        for(int i=1; i<=H; i+=R){
            if((i-R)>0 && (i+R)<=H){
                drawRing(arr,H,W,R,R+2,i,1,r,g,b);
                drawRing(arr,H,W,R,W-R-1,i,1,r,g,b);
            }
        }
        for(int i=1; i<=W; i+=R){
            if((i-R)>0 && (i+R)<=W){
                drawRing(arr,H,W,R,i,R+2,1,r,g,b);
                drawRing(arr,H,W,R,i,H-R-1,1,r,g,b);
            }
        }
    }
}

void turnImage(image *img, int grade, int x0, int y0, int x1, int y1){
    if(x0>x1 || y0>y1 || x0<=0 || y0<=0 || x1>img->bmih.width || y1>img-
>bmih.height){
        printf("Error: Put incorrect meaning of coordinates.\n");
        return;
    }
    if(grade!=90 && grade!=180 && grade!=270 && grade!=0){
        printf("Error: Put incorrect angle.\n");
        return;
    }
    if(x0==1 && y0==1 && x1==img->bmih.width && y1==img->bmih.height){
        if(grade == 90){
            unsigned int h = img->bmih.width;
            unsigned int w = img->bmih.height;
            RGB **arr = malloc((h)*sizeof(RGB*));
            for(int i=0; i<h; i++){
                arr[i] = calloc(sizeof(RGB)*(w)+(4-(w*3)%4)%4,1);
            }
            for(int i=0; i<img->bmih.height; i++){
                for(int j=0; j<img->bmih.width; j++){
                            drawPoint(&(arr[img->bmih.width-1-j][i]),img-
>pxlsArr[i][j].R,
                              img->pxlsArr[i][j].G,img->pxlsArr[i][j].B);
                }
            }
            for(int i=0; i<img->bmih.height; i++){
                free(img->pxlsArr[i]);
```

```
            }
            free(img->pxlsArr);
            img->bmih.height = h;
            img->bmih.width = w;
            img->pxlsArr = arr;
        } else if(grade == 180){
            RGB **arr = malloc((img->bmih.height)*sizeof(RGB*));
            for(int i=0; i<img->bmih.height; i++){
                  arr[i] = calloc(sizeof(RGB)*(img->bmih.width)+(4-(img-
>bmih.width*3)%4)%4,1);
            }
            for(int i=0; i<img->bmih.height; i++){
                for(int j=0; j<img->bmih.width; j++){
                            drawPoint(&(arr[img->bmih.height-1-i][j]),img-
>pxlsArr[i][j].R,
                               img->pxlsArr[i][j].G,img->pxlsArr[i][j].B);
                }
            }
            for(int i=0; i<img->bmih.height; i++){
                free(img->pxlsArr[i]);
            }
            free(img->pxlsArr);
            img->pxlsArr = arr;
        } else if(grade == 270){
            unsigned int h = img->bmih.width;
            unsigned int w = img->bmih.height;
            RGB **arr = malloc((h)*sizeof(RGB*));
            for(int i=0; i<h; i++){
                arr[i] = calloc(sizeof(RGB)*(w)+(4-(w*3)%4)%4,1);
            }
            for(int i=0; i<img->bmih.height; i++){
                for(int j=0; j<img->bmih.width; j++){
                            drawPoint(&(arr[j][img->bmih.height-1-i]),img-
>pxlsArr[i][j].R,
                               img->pxlsArr[i][j].G,img->pxlsArr[i][j].B);
                }
            }
            for(int i=0; i<img->bmih.height; i++){
                free(img->pxlsArr[i]);
            }
            free(img->pxlsArr);
            img->bmih.height = h;
            img->bmih.width = w;
            img->pxlsArr = arr;
        }
    } else {
        if(grade==90) {
            RGB **arr = malloc(img->bmih.height * sizeof(RGB *));
            for (int i = 0; i < img->bmih.height; i++) {
                  arr[i] = malloc(img->bmih.width * sizeof(RGB) + ((img-
>bmih.width * 3) % 4) % 4);
                for (int j = 0; j < img->bmih.width; j++) {
                      drawPoint(&(arr[i][j]), img->pxlsArr[i][j].R, img-
>pxlsArr[i][j].G, img->pxlsArr[i][j].B);
                }
            }
            int start = (int) (img->bmih.height - y1);
            int finish = (int) (img->bmih.height - y0);
```

```
            for (int i = start; i <= finish; i++) {
                for (int j = x0 - 1; j <= (x1 - 1); j++) {
                    drawPoint(&(arr[i][j]), 255, 255, 255);
                }
            }
            if (y1 - y0 >= x1 - x0) {
                 for (int i = 0; i <= img->bmih.width - x0 && i <= finish
- start; i++) {
                    for (int j = 0; j <= x1 - x0; j++) {
                          drawPoint(&(arr[finish - j][i + x0 - 1]), img-
>pxlsArr[i + start][x0 + j - 1].R,
                                  img->pxlsArr[i + start][x0 + j - 1].G,
img->pxlsArr[i + start][x0 + j - 1].B);
                    }
                }
            } else {
                for (int i = 0; i <= finish - start; i++) {
                    for (int j = 0; j <= x1 - x0 && j <= finish; j++) {
                          drawPoint(&(arr[finish - j][i + x0 - 1]), img-
>pxlsArr[i + start][x0 + j - 1].R,
                                  img->pxlsArr[i + start][x0 + j - 1].G,
img->pxlsArr[i + start][x0 + j - 1].B);
                    }
                }
            }
            for(int i=0; i<img->bmih.height; i++){
                free(img->pxlsArr[i]);
            }
            free(img->pxlsArr);
            img->pxlsArr = arr;
        } else if(grade==180) {
            RGB **arr = malloc(img->bmih.height * sizeof(RGB *));
            for (int i = 0; i < img->bmih.height; i++) {
                  arr[i] = malloc(img->bmih.width * sizeof(RGB) + ((img-
>bmih.width * 3) % 4) % 4);
                for (int j = 0; j < img->bmih.width; j++) {
                       drawPoint(&(arr[i][j]), img->pxlsArr[i][j].R, img-
>pxlsArr[i][j].G, img->pxlsArr[i][j].B);
                }
            }
            for (int i = y0; i <= y1; i++) {
                for (int j = x0; j <= x1; j++) {
                     drawPoint(&(arr[img->bmih.height-i][j-1]), 255, 255,
255);
                }
            }
            for(int i=y0; i<=y1; i++){
                for(int j=x0; j<=x1; j++){
                    drawPoint(&(arr[img->bmih.height-i][j-1]),
                                img->pxlsArr[img->bmih.height-(y0+y1-i)]
[x1+x0-j].R,
                                img->pxlsArr[img->bmih.height-(y0+y1-i)]
[x1+x0-j].G,
                                img->pxlsArr[img->bmih.height-(y0+y1-i)]
[x1+x0-j].B);
                }
            }
            for(int i=0; i<img->bmih.height; i++){
```

```
                free(img->pxlsArr[i]);
            }
            free(img->pxlsArr);
            img->pxlsArr = arr;
        } else if(grade==270){
            RGB **arr = malloc(img->bmih.height * sizeof(RGB *));
            for (int i = 0; i < img->bmih.height; i++) {
                  arr[i] = malloc(img->bmih.width * sizeof(RGB) + ((img-
>bmih.width * 3) % 4) % 4);
                for (int j = 0; j < img->bmih.width; j++) {
                      drawPoint(&(arr[i][j]), img->pxlsArr[i][j].R, img-
>pxlsArr[i][j].G, img->pxlsArr[i][j].B);
                }
            }

            for (int i = y0; i <= y1; i++) {
                for (int j = x0; j <= x1; j++) {
                     drawPoint(&(arr[img->bmih.height-i][j-1]), 255, 255,
255);
                }
            }
            if(y1 - y0 >= x1 - x0){
                int start = (int) (img->bmih.height - y1);
                int finish = (int) (img->bmih.height - y0);
                for(int i=0; i<=finish-start && i<=x1-1; i++){
                    for(int j=0; j<=x1-x0; j++){
                                drawPoint(&(arr[finish-j][x1-i-1]), img-
>pxlsArr[i + start][x1-j-1].R,
                                  img->pxlsArr[i + start][x1-j-1].G, img-
>pxlsArr[i + start][x1-j-1].B);
                    }
                }
                for(int i=0; i<img->bmih.height; i++){
                    free(img->pxlsArr[i]);
                }
                free(img->pxlsArr);
                img->pxlsArr = arr;
            } else {
                int start = (int) (img->bmih.height - y1);
                int finish = (int) (img->bmih.height - y0);
                for(int i=0; i<=finish-start && i<=x1-1; i++){
                    for(int j=0; j<=x1-x0 && j<=finish; j++){
                                drawPoint(&(arr[finish-j][x1-i-1]), img-
>pxlsArr[i + start][x1-j-1].R,
                                  img->pxlsArr[i + start][x1-j-1].G, img-
>pxlsArr[i + start][x1-j-1].B);
                    }
                }
                for(int i=0; i<img->bmih.height; i++){
                    free(img->pxlsArr[i]);
                }
                free(img->pxlsArr);
                img->pxlsArr = arr;
            }
        }
    }
    if(grade==0){
        return;
```

```
    }
}

void printHelp(){
     printf("bmpformat - program that supports CLI and works with bmp
format files;\n");
        printf("Only bmp files version 3 are supported (40 bytes
BITMAPINFOHEADER);\n");
    printf("bmp files with color table AREN'T supported;\n");
    printf("Image depth = 24 bites per pixel, without compression; number
of planes = 1;\n");
     printf("\tFormat of input: ./bmpformat [instruction] [name of input
file] -[key1]/--[long_ke1] [arg1] ... [name of output file] (the last
argument)\n\n");

    printf("\tInstructions:\n\n");

      printf("\t--DrawRectangle/-R [name of file] - drawing filled or
unfilled rectangle:\n");
         printf("\t\t-l/--leftcorner [x0.y0]- left corner coordinates of
rectangle.\n");
        printf("\t\t-r/--rightcorner [x1.y1]- right corner coordinates of
rectangle.\n");
        printf("\t\t-w/--width [number] - width of rectangle's frame (==0
by default).\n");
         printf("\t\t-c/--color1 [255.255.255] - color of frame if format
RGB (== 0.0.0 by default).\n");
         printf("\t\t-f/--filling [0/1] - fill, if there is, then 1, else
- 0 (==0 by default).\n");
           printf("\t\t-C/--color2 [255.255.255] - color of filling in
format RGB (== 0.0.0 by default).\n");
    printf("\t--Frame/-F [name of file] - drawing of frame:\n");
          printf("\t\t-s/--style [1/2/3/4] - choose the style of frame:\
n");
              printf("\t\t\t1 - double frame; 2 - frame in points; 3 -
stripes; 4 - rings;\n");
         printf("\t\t-c/--color [255.255.255] - color of frame in format
RGB (== 0.0.0 by default).\n");
           printf("\t\t-w/--width [number] - width of frame (== 0 by
default).\n");
     printf("\t--TurnImage/-T [name of file] - rotate the image by the
specified angle:\n");
         printf("\t\t-l/--leftcorner [x0.y0]- left corner coordinates of
zone (== 1.1 by default).\n");
        printf("\t\t-r/--rightcorner [x1.y1]- right corner coordinates of
zone (== width.height by default).\n");
        printf("\t\t-a/--angle [0/90/180/270] - angle (== 0 by default).\
n");
    printf("\t--DrawCircle/-L [name of file] - draw a filled or unfilled
circle:\n");
         printf("\t\t-p/--coordinates [x0.y0.R/x0.y0.x1.y1] - coordinates
of circle: coordinates of centre and radius or\n");
             printf("\t\t\tthe upper left and lower right corners of the
square where the circle id inscribed.\n");
           printf("\t\t-w/--width [number] width of the ring (== 0 by
default).\n");
         printf("\t\t-c/--color1 [255.255.255] - color of ring in format
RGB (== 0.0.0 by default).\n");
```

```
            printf("\t\t-f/-filling [0/1] - fill the circle - 1-filled
circle; 0-unfilled (== 0 by default).\n");
            printf("\t\t-C/--color2 [255.255.255] - color of filling in
format RGB (== 0.0.0 by default).\n");
    printf("\t--info/-i [name of file] - information about file.\n");
    printf("\t-h/--help/-? - about program.\n");
}

int main(int argc, char *argv[]) {

    if (argc < 3) { //Error
          if (argc>1 && (!strcmp(argv[1], "--help") || !strcmp(argv[1],
"-?") || !strcmp(argv[1], "-h")) ){
            printHelp();
            return 0;
        }
        printf("Error:\tToo few arguments was got by program!\n");
        printHelp();
        return 0;
    }
    else if (argc == 3) { //input and output file
        image img;
        if (readFile(argv[1], &img)) {
            printf("Try again!\n");
            printHelp();
            return 0;
        }
        writeFile(argv[2], &img);
        return 0;
    } else {
         if (!strcmp(argv[1], "--help") || !strcmp(argv[1], "-?") || !
strcmp(argv[1], "-h")) { //help
            image img;
            if (readFile(argv[2], &img)) {
                printf("Try again!\n");
                printHelp();
                return 0;
            }
            writeFile(argv[3], &img);
            printHelp();
            return 0;
        }
           if (!strcmp(argv[1], "--info") || !strcmp(argv[1], "-i")) {
//info
            image img;
            if (readFile(argv[2], &img)) {
                printf("Try again!\n");
                printHelp();
                return 0;
            }
            writeFile(argv[3], &img);
            printFileHeader(img.bmfh);
            printInfoHeader(img.bmih);
            return 0;
        }
         if (!strcmp(argv[1], "--DrawRectangle") || !strcmp(argv[1], "--
R")) { //draw Rectangle
            image img;
```

```
            if (readFile(argv[2], &img)) {
                printf("Try again!\n");
                printHelp();
                return 0;
            }

            struct option longOpt[] = {
                    {"leftcorner",required_argument ,NULL,'l'},
                    {"rightcorner",required_argument ,NULL,'r'},
                    {"width",required_argument,NULL,'w'},
                    {"color1",required_argument,NULL,'c'},
                    {"filling",required_argument,NULL,'f'},
                    {"color2",required_argument,NULL,'C'},
                    {"DrawRectangle",required_argument,NULL,'R'},
                    { NULL, 0, NULL, 0}
                    };
            char *opts = "l:r:w:c:f:C:R";
            int opt;
            int longIndex;
            int x0=-1,y0=-1,x1=-1,y1=-1,k;
            int w=0;
            int rl=0,gl=0,bl=0,f=0,rf=0,gf=0,bf=0;
            char *output_str = argv[argc - 1];
            opt = getopt_long(argc,argv,opts,longOpt,&longIndex);
            while(opt!=-1){
                switch(opt){
                    case 'l':
                        if(!optarg){
                            break;
                        }
                        k = sscanf(optarg,"%d.%d",&x0,&y0);
                        if (k!=2){
                               printf("Error: data could not be counted.\
n");
                            writeFile(output_str, &img);
                            return 0;
                        }
                        break;
                    case 'r':
                        if(!optarg){
                            break;
                        }
                        k = sscanf(optarg,"%d.%d",&x1,&y1);
                        if (k!=2){
                               printf("Error: data could not be counted.\
n");
                            writeFile(output_str, &img);
                            return 0;
                        }
                        break;
                    case 'w':
                        if(!optarg){
                            break;
                        }
                        k = sscanf(optarg,"%d",&w);
                        if (k!=1){
                            w = 0;
                        }
```

```
                        break;
                    case 'c':
                        if(!optarg){
                            break;
                        }
                        k = sscanf(optarg,"%d.%d.%d",&rl,&gl,&bl);
                        if (k!=3){
                            rl = 0;
                            gl = 0;
                            bl = 0;
                        }
                         if(rl>=0 && rl<=255 && gl>=0 && gl<=255 && bl>=0
&& bl<=255){
                            break;
                        } else {
                            printf("Error: invaluable colors.\n");
                            writeFile(output_str, &img);
                            return 0;
                        }
                    case 'f':
                        if(!optarg){
                            break;
                        }
                        k = sscanf(optarg,"%d",&f);
                        if (k!=1){
                            f = 0;
                        }
                        break;
                    case 'C':
                        if(!optarg){
                            break;
                        }
                        k = sscanf(optarg,"%d.%d.%d",&rf,&gf,&bf);
                        if (k!=3){
                            rf = 0;
                            gf = 0;
                            bf = 0;
                        }
                         if(rf>=0 && rf<=255 && gf>=0 && gf<=255 && bf>=0
&& bf<=255){
                            break;
                        } else {
                            printf("Error: invaluable colors.\n");
                            writeFile(output_str, &img);
                            return 0;
                        }
                    case 'R':
                        break;
                }
                opt = getopt_long(argc,argv,opts,longOpt,&longIndex);
            }
            if(x1>0 && y1>0 && x0>0 && y1>0){
                if(f){
                     drawFilledRectangle(img.pxlsArr,(int)img.bmih.width,
(int)img.bmih.height,x0,y0,x1,y1,w,rl,gl,bl,rf,gf,bf);
                } else {
                            drawRectangle(img.pxlsArr,(int)img.bmih.width,
(int)img.bmih.height,x0,y0,x1,y1,w,rl,gl,bl);
```

```
                }
            } else {
                printf("Error: Program didn't get coordinates.\n");
            }
            writeFile(output_str, &img);
            return 0;
        } else if(!strcmp(argv[1],"--Frame") || !strcmp(argv[1],"-F")){
            image img;
            if (readFile(argv[2], &img)) {
                printf("Try again!\n");
                printHelp();
                return 0;
            }
            struct option longOpt[] = {
                    {"style",required_argument ,NULL,'s'},
                    {"width",required_argument,NULL,'w'},
                    {"color",required_argument,NULL,'c'},
                    {"Frame",required_argument,NULL,'F'},
                    { NULL, 0, NULL, 0}
            };
            char *opts = "s:c:w:F";
            int opt;
            int longIndex;
            int style = 0;
            int r=0,g=0,b=0,w=0;
            char *output_str = argv[argc - 1];
            int k;
            opt = getopt_long(argc,argv,opts,longOpt,&longIndex);
            while(opt!=-1){
                switch(opt){
                    case 's':
                        if(!optarg){
                            break;
                        }
                        k = sscanf(optarg,"%d",&style);
                        if (k!=1){
                               printf("Error: data could not be counted.\
n");
                            writeFile(output_str, &img);
                            return 0;
                            } else if(style!=1 && style!=2 & style!=3 &&
style!=4) {
                            printf("Error: invaluable style.\n");
                            return 0;
                        }
                        break;
                    case 'w':
                        if(!optarg){
                            break;
                        }
                        k = sscanf(optarg,"%d",&w);
                        if (k!=1){
                            w = 0;
                        }
                        break;
                    case 'c':
                        if(!optarg){
                            break;
```

```
                        }
                        k = sscanf(optarg,"%d.%d.%d",&r,&g,&b);
                        if (k!=3){
                            r = 0;
                            g = 0;
                            b = 0;
                        }
                          if(r>=0 && r<=255 && g>=0 && g<=255 && b>=0 &&
b<=255){
                            break;
                        } else {
                            printf("Error: invaluable colors.\n");
                            writeFile(output_str, &img);
                            return 0;
                        }
                    case 'F':
                        break;
                }
                opt = getopt_long(argc,argv,opts,longOpt,&longIndex);
            }
            switch(style){
                case 1:
                                drawFrame1(img.pxlsArr,(int)img.bmih.height,
(int)img.bmih.width,1,1,(int)img.bmih.width,
                                  (int)img.bmih.height,w,r,g,b);
                    break;
                case 2:
                                drawFrame2(img.pxlsArr,(int)img.bmih.height,
(int)img.bmih.width,w,r,g,b);
                    break;
                case 3:
                                drawFrame3(img.pxlsArr,(int)img.bmih.height,
(int)img.bmih.width,w,r,g,b);
                    break;
                case 4:
                                drawFrame4(img.pxlsArr,(int)img.bmih.height,
(int)img.bmih.width,w,r,g,b);
                    break;
                case 0:
                    printf("Error: Style wasn't got.\n");
                    break;
            }
            writeFile(output_str, &img);
            return 0;
          } else if(!strcmp(argv[1],"--TurnImage") || !strcmp(argv[1],"-
T")){
            image img;
            if (readFile(argv[2], &img)) {
                printf("Try again!\n");
                printHelp();
                return 0;
            }
            struct option longOpt[] = {
                    {"leftcorner",required_argument,NULL,'l'},
                    {"rightcorner",required_argument,NULL,'r'},
                    {"angle",required_argument,NULL,'a'},
                    {"TurnImage",required_argument,NULL,'T'},
                    { NULL, 0, NULL, 0}
```

```
            };
            char *opts = "l:r:a:T";
            int opt;
            int longIndex;
            char *output_str = argv[argc - 1];
            int k,x0=-1,y0=-1,x1=-1,y1=-1,ang;
            opt = getopt_long(argc,argv,opts,longOpt,&longIndex);
            while(opt!=-1){
                switch(opt){
                    case 'l':
                        if(!optarg){
                            break;
                        }
                        k = sscanf(optarg,"%d.%d",&x0,&y0);
                        if (k!=2){
                               printf("Error: data could not be counted.\
n");
                            writeFile(output_str, &img);
                            return 0;
                        }
                        break;
                    case 'r':
                        if(!optarg){
                            break;
                        }
                        k = sscanf(optarg,"%d.%d", &x1,&y1);
                        if (k!=2){
                               printf("Error: data could not be counted.\
n");
                            writeFile(output_str, &img);
                            return 0;
                        }
                        break;
                    case 'a':
                        if(!optarg){
                            break;
                        }
                        k = sscanf(optarg,"%d",&ang);
                        if (k!=1){
                            ang=0;
                        }
                        break;
                    case 'T':
                        break;
                }
                opt = getopt_long(argc,argv,opts,longOpt,&longIndex);
            }
            if(x1>0 && y1>0 && x0>0 && y1>0){
                turnImage(&img,ang,x0,y0,x1,y1);
            } else {
                printf("Error: Program didn't get coordinates.\n");
            }
            writeFile(output_str, &img);
            return 0;
        } else if(!strcmp(argv[1],"--DrawCircle") || !strcmp(argv[1],"-
L")){
            image img;
            if (readFile(argv[2], &img)) {
```

```
                printf("Try again!\n");
                printHelp();
                return 0;
            }
            struct option longOpt[] = {
                    {"coordinates",required_argument,NULL,'p'},
                    {"width",required_argument,NULL,'w'},
                    {"color1",required_argument,NULL,'c'},
                    {"color2",required_argument,NULL,'C'},
                    {"filling",required_argument,NULL,'f'},
                    {"DrawCircle",required_argument,NULL,'L'},
                    { NULL, 0, NULL, 0}
            };
            char *opts = "p:w:c:C:f:L";
            int opt;
            int longIndex;
            char *output_str = argv[argc - 1];
            int x0,y0,R,x1,y1,w=0,rl=0,gl=0,bl=0,f=0,rf=0,gf=0,bf=0,k;
            opt = getopt_long(argc,argv,opts,longOpt,&longIndex);
            while(opt!=-1){
                switch(opt){
                    int tmp1,tmp2,tmp3,tmp4;
                    case 'p':
                        if(!optarg){
                            break;
                        }
                                    k = sscanf(optarg, "%d.%d.%d.%d",
&tmp1,&tmp2,&tmp3,&tmp4);
                        if(k==4){
                            x0=tmp1; y0=tmp2; x1=tmp3; y1=tmp4;
                            break;
                        } else if(k==3){
                            x0=tmp1; y0=tmp2; R=tmp3;
                            break;
                        } else {
                               printf("Error: data could not be counted.\
n");
                            writeFile(output_str, &img);
                            return 0;
                        }
                        break;
                    case 'w':
                        if(!optarg){
                            break;
                        }
                        k = sscanf(optarg,"%d",&w);
                        if(k!=1){
                            w=0;
                        }
                        break;
                    case 'f':
                        if(!optarg){
                            break;
                        }
                        k = sscanf(optarg,"%d",&f);
                        if(k!=1){
                            f=0;
                        }
```

```
                        break;
                    case 'c':
                        if(!optarg){
                            break;
                        }
                        k = sscanf(optarg,"%d.%d.%d",&rl,&gl,&bl);
                        if (k!=3){
                            rl = 0;
                            gl = 0;
                            bl = 0;
                        }
                         if(rl>=0 && rl<=255 && gl>=0 && gl<=255 && bl>=0
&& bl<=255){
                            break;
                        } else {
                            printf("Error: invaluable colors.\n");
                            writeFile(output_str, &img);
                            return 0;
                        }
                    case 'C':
                        if(!optarg){
                            break;
                        }
                        k = sscanf(optarg,"%d.%d.%d",&rf,&gf,&bf);
                        if (k!=3){
                            rf = 0;
                            gf = 0;
                            bf = 0;
                        }
                         if(rf>=0 && rf<=255 && gf>=0 && gf<=255 && bf>=0
&& bf<=255){
                            break;
                        } else {
                            printf("Error: invaluable colors.\n");
                            writeFile(output_str, &img);
                            return 0;
                        }
                    case 'L':
                        break;
                }
                opt = getopt_long(argc,argv,opts,longOpt,&longIndex);
            }
            if(x1>0 && y1>0 && x0>0 && y1>0){
                if(f){
                                drawCircle(img.pxlsArr,(int)img.bmih.height,
(int)img.bmih.width,(x1-x0)/2,
                    x0+ (x1-x0)/2,y0 + (y1-y0)/2,w,rl,gl,bl,rf,gf,bf);
                } else {
                                  drawRing(img.pxlsArr,(int)img.bmih.height,
(int)img.bmih.width,
                    (x1-x0)/2,x0+ (x1-x0)/2,y0 + (y1-y0)/2,w,rl,gl,bl);
                }
            } else if(x0>0 && y0>0 && R>0){
                if(f){
                                drawCircle(img.pxlsArr,(int)img.bmih.height,
(int)img.bmih.width,R,x0,y0,w,rl,gl,bl,rf,gf,bf);
                } else {
```

```
                                drawRing(img.pxlsArr,(int)img.bmih.height,
(int)img.bmih.width,R,x0,y0,w,rl,gl,bl);
                }
            } else {
                printf("Error: Program didn't get coordinates.\n");
            }
            writeFile(output_str, &img);
            return 0;
        }
    }
}
```