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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка изображения

Студент гр. 3383 _____ Матвеев Н.С.

Преподаватель _____ Гаврилов А.В.

Санкт-Петербург

2024

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Матвеев Н.С.

Группа 3383

Тема работы: Обработка изображения

Вариант 9
Программа **обязательно должна иметь CLI** (опционально дополнительное использование GUI). Более подробно тут: http://se.moevm.info/doku.php/courses:programming:rules_extra_kurs
Программа должна реализовывать весь следующий функционал по обработке bmp-файла

Общие сведения

- 24 бита на цвет
- без сжатия
- файл может не соответствовать формату BMP, т.е. необходимо проверка на BMP формат (дополнительно стоит помнить, что версий у формата несколько). Если файл не соответствует формату BMP или его версии, то программа должна завершиться с соответствующей ошибкой.
- обратите внимание на выравнивание; мусорные данные, если их необходимо дописать в файл для выравнивания, должны быть нулями.
- обратите внимание на порядок записи пикселей
- все поля стандартных BMP заголовков в выходном файле должны иметь те же значения что и во входном (разумеется, кроме тех, которые должны быть изменены).

Программа должна иметь следующую функции по обработке изображений:

1. Рисование квадрата с диагоналями. Флаг для выполнения данной операции: `--squared_lines`. Квадрат определяется:
    - Координатами левого верхнего угла. Флаг `--left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y
    - Размером стороны. Флаг `--side_size`. На вход принимает число больше 0
    - Толщиной линий. Флаг `--thickness`. На вход принимает число больше 0
    - Цветом линий. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)
    - Может быть залит или нет (диагонали располагаются “поверх” заливки). Флаг `--fill`. Работает как бинарное значение: флага нет – false , флаг есть – true.
    - Цветом которым он залит, если пользователем выбран залитый. Флаг `--fill_color` (работает аналогично флагу `--color`)
2. Фильтр rgb-компонент. Флаг для выполнения данной операции: `--rgbfilter`. Этот инструмент должен позволять для всего изображения либо установить в диапазоне от 0 до 255 значение заданной компоненты. Функционал определяется
    - Какую компоненту требуется изменить. Флаг `--component_name`. Возможные значения `red`, `green` и `blue`.
    - В какой значение ее требуется изменить. Флаг `--component_value`. Принимает значение в виде числа от 0 до 255
3. Поворот изображения (части) на 90/180/270 градусов. Флаг для выполнения данной операции: `--rotate`. Функционал определяется
    - Координатами левого верхнего угла области. Флаг `--left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y

- Координатами правого нижнего угла области. Флаг `--right_down`, значение задаётся в формате `right.down`, где right – координата по x, down – координата по y
- Углом поворота. Флаг `--angle`, возможные значения: `90`, `180`, `270`

4. Рисование окружности. Флаг для выполнения данной операции: `--circle`. Окружность определяется:

- координатами ее центра и радиусом. Флаги `--center` и `--radius`. Значение флаг `--center` задаётся в формате `x.y`, где x – координата по оси x, y – координата по оси y. Флаг `--radius` На вход принимает число больше 0
- толщиной линии окружности. Флаг `--thickness`. На вход принимает число больше 0
- цветом линии окружности. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)
- окружность может быть залитой или нет. Флаг `--fill`. Работает как бинарное значение: флага нет – false , флаг есть – true.
- цветом которым залита сама окружность, если пользователем выбрана залитая окружность. Флаг `--fill_color` (работает аналогично флагу `--color`)

Каждую подзадачу следует вынести в отдельную функцию, функции сгруппировать в несколько файлов (например, функции обработки текста в один, функции ввода/вывода в другой). Сборка должна осуществляться при помощи make и Makefile или другой системы сборки.

Содержание пояснительной записки:

«Аннотация», «Содержание», «Введение», «Заключение», «Приложение А», «Приложение Б», «Список использованных источников».

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 2_.05.2024

Дата защиты реферата: 29.05.2024

Студент _____ Матвеев Н.С.

Преподаватель _____ Гаврилов А.В.

АННОТАЦИЯ

Курсовая работа представляет собой разработку консольного приложения на языке Си, выполняющего обработку изображения формата BMP. Программа собрана с помощью Makefile.

Взаимодействие с программой происходит при помощи CLI(Command Line Interface).

В программе реализовано 4 функции для обработки изображения: рисование квадрата с диагоналями, рисование окружности, фильтр rgb-компонент и поворот изображения(части) на 90, 180 или 270 градусов. Помимо этого, можно вызвать информацию о файле и справку о работе с программой.

SUMMARY

The course work is the development of a console application in the C language that performs image processing in the BMP format. The program is built using a Makefile.

Interaction with the program takes place using the CLI (Command Line Interface).

The program implements 4 functions for image processing: drawing a square with diagonals, drawing a circle, an rgb component filter and rotating the image (part) by 90, 180 or 270 degrees. In addition, you can call up information about the file and help with the program.

СОДЕРЖАНИЕ

	Введение	8
1.	Заголовочный файл BmpRead.h	9
1.1.	Используемые библиотеки	9
1.2.	Структура BmpFileHeader	9
1.3.	Структура BmpInfoHeader	9
1.4	Структура Rgb	9
2.	Работа с BMP-файлом	10
2.1.	Функция считывания и записи	10
2.2.	Функция вывода информации об изображении	10
2.3	Функция освобождения памяти	10
3.	Обработка BMP-файла	11
3.1.	Функция рисования квадрата	11
3.2.	Функция фильтр-rgb	12
3.3	Функция поворота	12
3.4	Функция рисования окружности	13
4.	Интерфейс командной строки	15
4.1.	Getopt и функция main	15
4.2.	Функция вывода справки	16
5.	Сборка проекта	17
	Заключение	18
	Список использованных источников	19
	Приложение А. Примеры работы программы	20
	Приложение В. Исходный код программы	28

ВВЕДЕНИЕ

Цель работы: изучить формат изображения BMP, научиться работать с BMP файлами на языке Си, создать консольное приложение с поддержкой CLI для обработки изображений формата BMP.

Задачи:

- Изучить BMP формат и особенности при работе с ним.
- Реализовать CLI.
- Реализовать функции обработки BMP файла.
- Обработать исключительные случаи.
- Протестировать программу на различных изображениях.
- Собрать программу при помощи Makefile.

1. ЗАГОЛОВОЧНЫЙ ФАЙЛ BMPREAD.H

1.1. Используемые библиотеки

- getopt.h - Эта библиотека содержит функции для разбора параметров командной строки.
- ctype.h - предоставляет функции для работы с символьными данными.
- stdio.h - предоставляет функции для ввода/вывода информации.
- stdlib.h - содержит функции для управления памятью, преобразований строк в числа и обратно.
- string.h - содержит функции для работы с символьными строками.

1.2. Структура BmpFileHeader

Структура *BmpFileHeader* хранит информацию о файле и имеет 5 полей: *signature*(2 байта), *filesize* (4 байта), *reserved1* (2 байта), *reserved2* (2 байта), *pixelArrOffset* (4 байта).

1.3. Структура BmpInfoHeader

Структура *BmpInfoHeader* хранит информацию о BMP и имеет 11 полей: size (4 байта), width (4 байта), height (4 байта), planes (2 байта), bit_count(4 байта), compression (4 байта), image_size (4 байта), x_resolution (4 байта), y_resolution (4 байта), colors (4 байта), important_colors (4 байта).

1.4. Структура Rgb

Структура Rgb хранит значения красной, зеленой и синей компоненты цвета пикселя (1 байт).

Для корректного считывания данных из файла в реализованные структуры необходимо выполнить выравнивание при помощи *pragma pack*.

2. РАБОТА С BMP-ФАЙЛОМ

2.1. Функция считывания и записи

Расположение: BmpRead.c

Эта функция открывает указанный BMP-файл для считывания, используя функцию *fopen()* с типом доступа r (чтение файла). Присутствует обработка ошибок: если файл невозможно открыть, то программа завершает свою работу с выводом сообщения об ошибке; если файл не формата BMP (происходит проверка при помощи *strcmp()* - сравнивается поле *signature* с типом BM), то программа завершает свою работу с выводом ошибки; если файл сжат (поле compression !=0), то программа завершает свою работу с выводом ошибки; если глубина цвета не соответствует ожидаемой(поле *bitsPerPixel* != 24), то программа завершает свою работу с выводом ошибки.

Структуры *BmpInfoHeader* и *BmpFileHeader* считываются при помощи *fread()*. Для считывания пикселей выделяется память под двумерный массив при помощи *malloc()*, считываются пиксели построчно при помощи цикла *for* и функции *fread()*. Размер под одну строку определяется значениями ширины и выравнивая. Выравнивание корректирует набор пикселей в одной строке до размера кратного 4.

Функция возвращает считанный массив пикселей.

2.2. Функция вывода информации об изображении

Расположение: BmpRead.c

Функция выводит в консоль информацию о считанном файле(все данные, которые содержатся в структурах *BmpInfoHeader* и *BmpFileHeader*) при помощи *printf()*.

2.3. Функция освобождения памяти

Расположение: BmpRead.c

Функция освобождает память, выделенную под структуры *BmpInfoHeader* и *BmpFileHeader* и выделенную под массив пикселей при помощи функции *free()*.

3. ОБРАБОТКА BMP-ФАЙЛА

3.1. Функция рисования квадрата

Расположение: BmpDraw.c

Функция рисует квадрат с диагоналями. На вход получает: массив пикселей, цвет, толщину, длину стороны, координаты левого верхнего угла, флаг заливки(0 или 1), цвет заливки — если флаг равен 1. Происходит проверка координат. В случае чего появляются переменные для регулирования границ. Высчитывается толщина внешняя и внутренняя (для рисования с одной стороны и другой у основной линии). Считывается цвет линий при помощи *sscanf().*

Если флаг заливки равен 1, то первоначально выполняется она. Считывается переданный цвет при помощи *sscanf()*. При помощи условных операторов проверяются границы заливки для регулировки координат. Заливка происходит при помощи циклов *for* и изменении компонентов цвета путем присваивания для каждого пикселя.

Сначала рисуются верхняя и нижняя сторона квадрата, сразу учитывая толщину(циклом *for* перебирается толщина). Если верхняя граница вне картинки, то сторона не рисуется (то же самое и с нижней стороной). Проверка происходит, используя условные операторы для переменных, регулирующие границы. Координата по у нижней стороны высчитывается как левая верхняя координата минус длина стороны.

Принцип рисования боковых сторон точно такой же. Циклом *for* перебирается толщина. Если левая граница вне картинки, то левая сторона не рисуется(то же самое с правой). Проверка происходит, используя условные операторы для переменных, регулирующие границы. Координата по х правой стороны высчитывается как левая верхняя координата плюс длина стороны.

Диагонали рисуются при помощи алгоритма Брезенхема. Высчитываются координаты для начало и конца каждой. Вводятся переменные специально для алгоритма. Первая диагональ рисуется снизу вверх, слева направо. Вторая

диагональ рисуется сверху вниз, слева направо. При этом циклом *for* перебирается толщина.

Функция ничего не возвращает.

3.2. Функция фильтр-rgb

Расположение: BmpChange.c

Функция меняет значение заданной компоненты цвета для всей картинки. На вход получает массив пикселей, цвет, новое значение для цвета, размер картинки. Смена компоненты происходит при помощи двух циклов *for* и условных операторов *if/else if*.

Функция ничего не возвращает.

3.3. Функция поворота

Расположение: BmpChange.c

Сначала обрабатываются все случаи с неправильным вводом координат или крайние случаи при помощи условных операторов(возможна смена координат правой с левой или если координаты вне картинки, то они меняются на граничные для данной картинки). Вводится переменная флаг, которая становится равной 1, когда требуется поворот на 90 или 270 градусов, при этом координаты такие, что необходимо переворачивать всю картинку.

Создается двумерный массив пикселей, для него выделяется память при помощи *malloc()* и цикла *for*. В нем будет хранится поворачиваемая область со считанной картинки.

Поворот на 180 осуществляется путем записи на место верхних строк - нижних(область переворачивается). Для этого используются 2 цикла *for* и созданный массив данной области(из него берутся нужные строки пикселей и заменяют ими строки в исходном массиве).

Поворот на 90 и 270 для всей картинки выполняется когда флаг равен 1. Для этого меняется размер исходного массива при помощи *realloc()* и цикла *for*. Меняются также исходные width и height. Поворот на 270: заполнение идет слева

направо и сверху вниз, по копии цикл *for* бежит снизу вверх и слева направо. Поворот на 90: заполнение идет слева направо и сверху вниз, по копии цикл *for* бежит сверху вниз и справа налево.

Поворот на 90 и 270 для части изображения выполняется относительно центра области. Высчитываются координаты левого верхнего угла и правого нижнего угла области, в которую необходимо вставить перевернутую область. Поворот на 270: заполнение идет слева направо и сверху вниз, по копии цикл *for* бежит снизу вверх и слева направо. Поворот на 90: заполнение идет слева направо и сверху вниз, по копии цикл *for* бежит сверху вниз и справа налево.

3.4. Функция рисования окружности

Расположение: BmpDraw.c

Функция рисует окружность.

На вход получает: массив пикселей, цвет, толщину, радиус, координаты центра, флаг заливки(0 или 1), цвет заливки — если флаг равен 1. Высчитывается толщина внешняя и внутренняя (для рисования с одной стороны и другой у основной линии). Считывается цвет и координаты центра при помощи функции *sscanf()*.

Если флаг заливки равен 1, то первоначально выполняется она. Считывается переданный цвет при помощи *sscanf()*. Область для заливки определяется квадратом(центр минус радиус во все четыре стороны). Внутри квадрата заливаются те пиксели, которые подходят под уравнение для точки внутри окружности. Заливка происходит при помощи циклов *for* и изменении компонентов цвета путем присваивания для каждого пикселя.

Циклом *for* перебирается толщина(нужна для изменения радиуса). Сначала рисуется полоска пикселей(соответствует толщине) над центром и под ним на расстоянии радиуса. Окружности рисуются при помощи математического алгоритма Брезенхема. Вводятся переменные специально для алгоритма.

Оставшиеся не закрашенные пиксели толщины обрабатываются отдельно. Создаются координаты квадратной области, которую занимает окружность.

Далее двумя циклами *for* перебираются пиксели данной области, если они находятся вне окружности с самым маленьким радиусом, но внутри окружности с самым большим радиусом, то пиксель закрашивается.

4. ИНТЕРФЕЙС КОМАНДНОЙ СТРОКИ

4.1. Getopt и функция main

Расположение: main.c.

В функции *main* происходит вся основная работа программы: ввод изображения, взаимодействие с флагами и аргументами командной строки, вызов функций обработки, вывод изображений. Подключены все требующиеся заголовочные файлы.

Getopt. Создается структура с длинными опциями, в которой описаны флаги: их названия, наличие аргумента и короткий флаг. Создается строка с короткими флагами. Для 4х функций, которые реализованы в программе созданы двумерные массивы, которые будут хранить аргументы для выполнения соответствующей функции. Для каждого массива выделяется память при помощи *malloc().* Создаются общие переменные для аргументов, которые одинаковые у разных функций.

При помощи цикла *while* и функции *getopt_long()* - обрабатываются команды, введенные пользователем в консоль. Для определения какой блок кода будет выполняться – используется оператор множественного выбора *switch case.* В зависимости от переданной команды обрабатывается определенный case. Внутри предусмотрены проверки на все крайние случаи и ошибки. Существует переменная флаг, в которую записывается номер функции, которую вызывает пользователь. Также внутри заполняется массив аргументов для данной функции.

В зависимости от значения в флаге вызывается функция, которую выполняет программа. Предусмотрены проверки на все крайние случаи и ошибки.

Изменное изображение нужно сохранить в новом файле. Для открытия используется функия *fopen()* с флагом wb – для записи в бинарном режиме. Записываются исходные стуктуры при помощи *fwrite()*. Далее записывается сам массив пикселей при помощи цикла *for* и *fwrite()* с учетом выравнивания.

В конце очищается память от созданных массивов для аргументов каждой функции, а также вызывается функция *free_bmp()* для очистки от считанного BMP файла.

4.2. Функция вывода справки

Расположение: consol.c

Функция выводит в консоль информацию о варианте курсовой работы и разработчике, также печатается справка по взаимодействию с программой (описываются все флаги, с которыми работает программа). Вывод осуществляется при помощи функции *printf().*

5. СБОРКА ПРОЕКТА

Для сборки проекта было решено использовать Makefile.

Программа разделена на цели:

- all – выполнение цели main и clean.
- main – сборка основной программы, получение исполняемого файла, результат целей main.o consol.o BmpRead.o BmpChange.o BmpDraw.o
- main.o – получение объектного файла main.c, заголовки consol.h BmpRead.h BmpChange.h BmpDraw.h
- consol.o – получение объектного файла consol.c, заголовки: consol.h, BmpRead.h
- BmpRead.o – получение объектного файла BmpRead.c, заголовки: BmpChange.h, BmpRead.h
- BmpDraw.o – получение объектного файла BmpDraw.c, заголовки: BmpDraw.h, BmpRead.h
- BmpChange.o – получение объектного файла BmpChange.c, заголовки: BmpChange.h, BmpRead.h
- clean — удаление всех объектных файлов и сам исполняемый файл cw, при помощи rm -rf.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были изучены: структура файла изображения типа BMP и особенности работы с ним на языке программирования Си, также были изучены: библиотека getopt,h для написания консольного приложения с Command Line Interface, математические алгоритмы построения прямой и окружности.

В результате написана консольная программа, способная считывать, записывать, обрабатывать изображение. Пользователь может выбрать одну из четырех операций: рисование квадрата с диагоналями, рисование окружности, поворот части изображения (на 90, 180, 270 градусов), rgb-фильтр.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Брайан Керниган, Деннис Ритчи — Язык программирования Си.

2. Описание методических указаний / сост.: М. М. Заславский, А. А. Лисс, А. В. Гаврилов и др. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2024. 36 с.

3. Сведения по getopt // курс по Linux на moevm.info. URL: https://e.moevm.info/mod/lesson/view.php?id=861&pageid=952&startlastseen=yes

ПРИЛОЖЕНИЕ А
ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

1. Справка о программе

   ./cw —help

```
nikita@nikita-Nitro-AN515-55:~/Проверка курсовой на компе$ ./cw --help
Course work for option 5.9, created by Nikita Matveev
Программа предназначена для обработки BMP изображений: без сжатия, 24 бита на пиксель.
Доступный функционал:
-h/--help  -  Вывод справки о программе
-I/--info  -  Вывод информации об изображении
-i/--input  -  Имя входного изображения
-o/--output  -  Имя выходного изображения
--squared_lines - Флаг для рисования квадрата с диагоналями. Квадрат определяется:
    --left_up  -  Координаты левого угла, значение задаётся в формате `left.up`, где left - координата по x, up - координата по y
    --side_size - Размер стороны, значение должно быть целым числом больше 0
    --thickness  -  Толщина линии, значение должно быть целым числом больше 0
    --color  -  Цвет линий, задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb - числа, задающие цветовую компоненту
    --fill - Наличие заливки, работает как бинарное значение: флага нет - false , флаг есть - true.
    --fill_color - Цвет заливки, задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb - числа, задающие цветовую компоненту
--rgbfilter - Флаг для изменения заданной компоненты цвета на переданное значение. Функционал определяется:
    --component_name - Имя компоненты, возможные значения `red`, `green` и `blue`
    --component_value - Принимает значение в виде числа от 0 до 255
--rotate - Флаг для поворота изображения (части) на 90/180/270 градусов. Функционал определяется:
    --left_up  -  Координаты левого угла, значение задаётся в формате `left.up`, где left - координата по x, up - координата по y
    --right_down  -  Координаты правого угла, значение задаётся в формате `right.down`, где right - координата по x, down - координата по y
    --angle - Угол поворота, допустимые значения: `90`, `180`, `270`
--circle - Флаг для рисования окружности. Окружность определяется:
    --center - Координаты центра, задаётся в формате `x.y`, где x - координата по оси x, y - координата по оси y.
    --radius - Радиус, значение должно быть целым числом больше 0
    --thickness  -  Толщина линии, значение должно быть целым числом больше 0
    --color  -  Цвет линий, задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb - числа, задающие цветовую компоненту
    --fill - Наличие заливки, работает как бинарное значение: флага нет - false , флаг есть - true.
    --fill_color - Цвет заливки, задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb - числа, задающие цветовую компоненту
```

2. Информация о файле

   ./cw --info /home/nikita/Загрузки/Lake.bmp

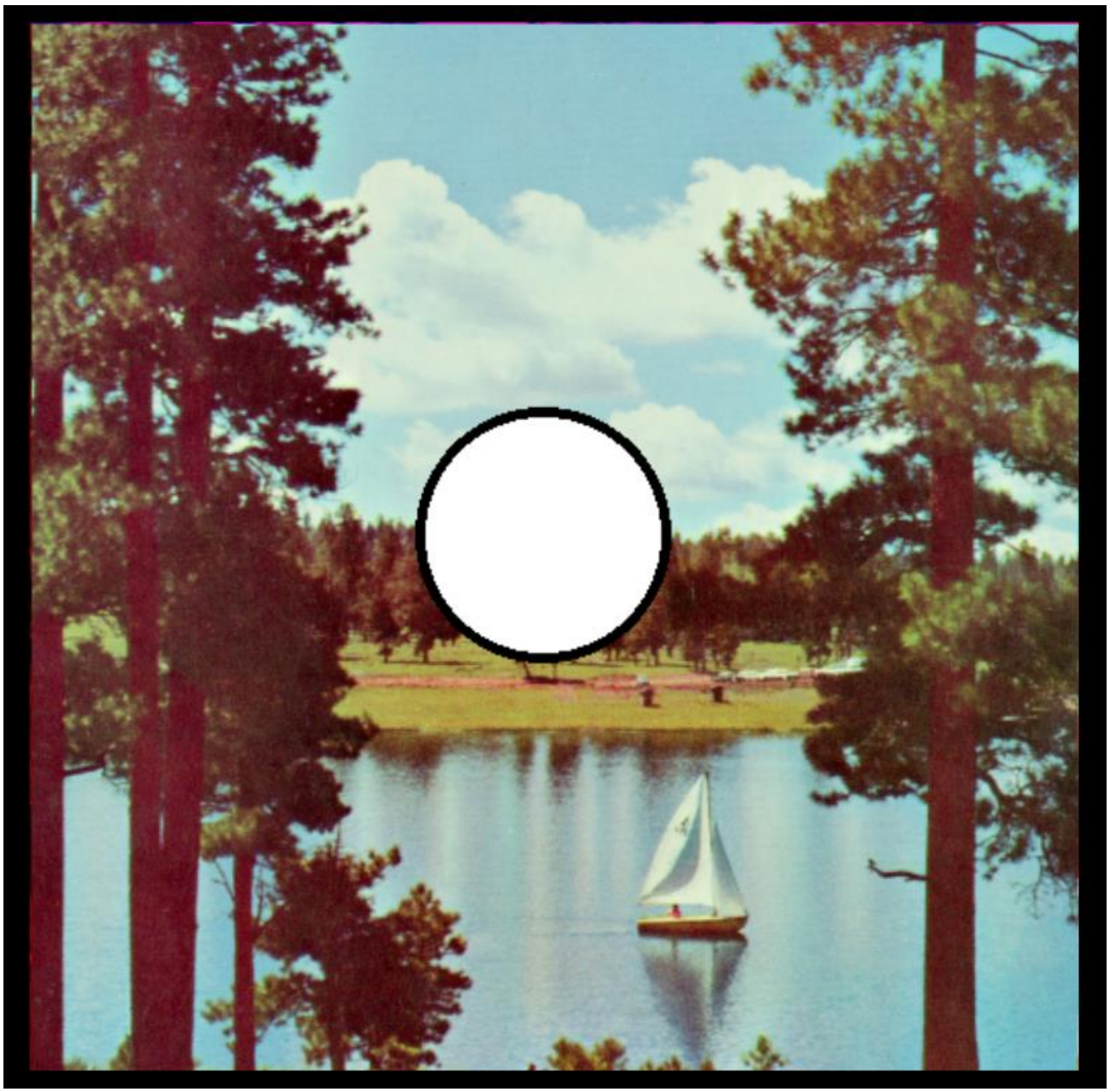
```
nikita@nikita-Nitro-AN515-55:~/Проверка курсовой на компе$ ./cw --info /home/nikita/Загрузки/Lake.bmp
Signature: BM
FileSize: 786486
PixelArrOffset: 54
HeaderSize: 40
Width: 512
Height: 512
Planes: 1
BitsPerPixel: 24
Compression: 0
ImageSize: 0
xPixelsPerMeter: 2834
yPixelsPerMeter: 2834
ColorsInColorTable: 0
ImportantColorCount: 0
nikita@nikita-Nitro-AN515-55:~/Проверка курсовой на компе$ ./cw -I /home/nikita/Загрузки/Lake.bmp
Signature: BM
FileSize: 786486
PixelArrOffset: 54
HeaderSize: 40
Width: 512
Height: 512
Planes: 1
BitsPerPixel: 24
Compression: 0
ImageSize: 0
xPixelsPerMeter: 2834
yPixelsPerMeter: 2834
ColorsInColorTable: 0
ImportantColorCount: 0
```

3. Рисование квадрата

./cw --squared_lines --side_size 50 --color 0.0.0 --thickness 4 --left_up 250.250 /home/nikita/Загрузки/Lake.bmp

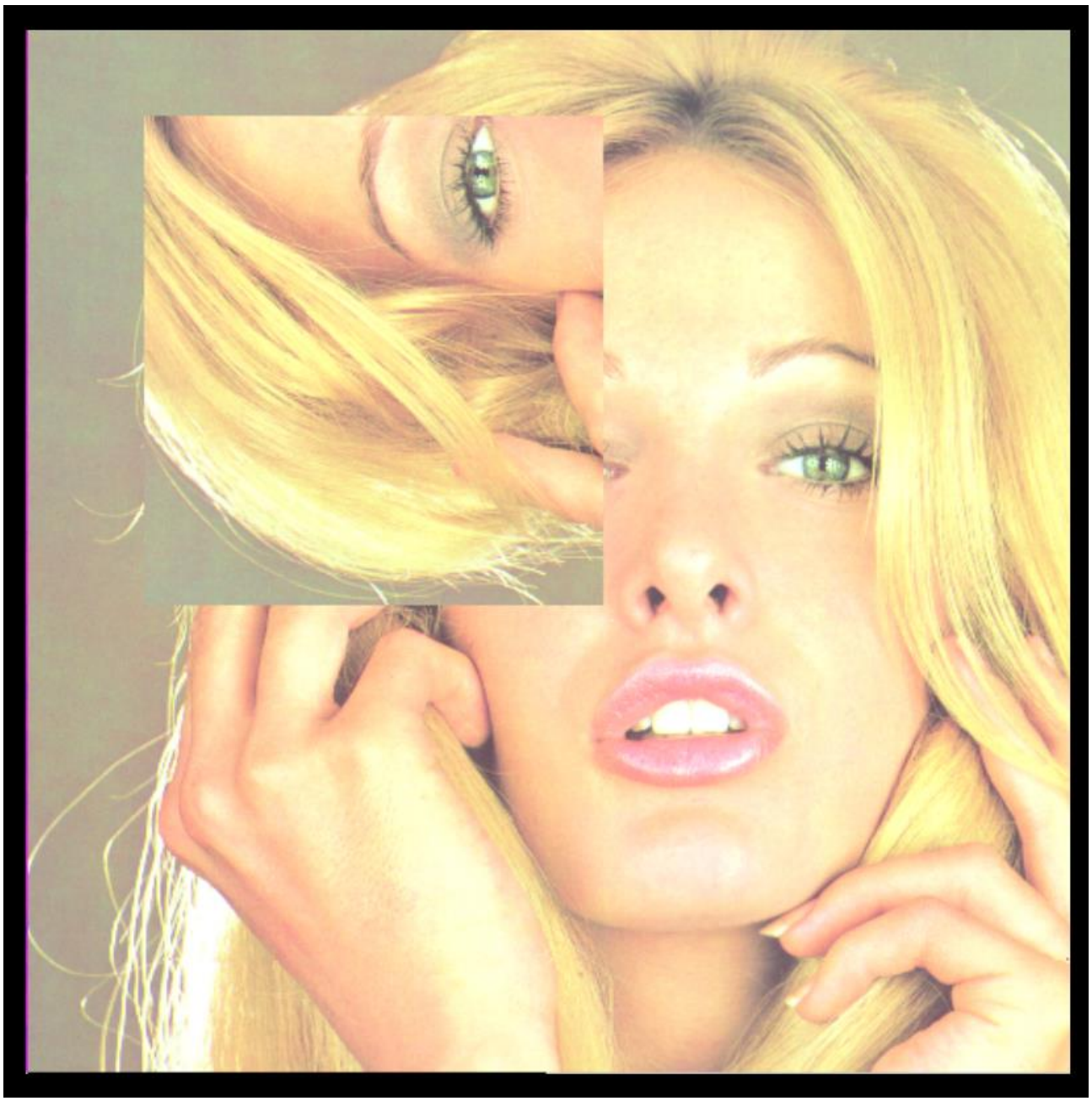
4. Рисование круга

./cw --input /home/nikita/Загрузки/Lake.bmp --circle --fill_color 255.255.255 --radius 60 --thickness 5 --fill --color 0.0.0 --center 250.250 --output ./outt.bmp



5. Функция поворота на 90

./cw --rotate --angle 90 --left_up 50.50 --right_down 290.275 /home/nikita/Загрузки/Tiffany.bmp



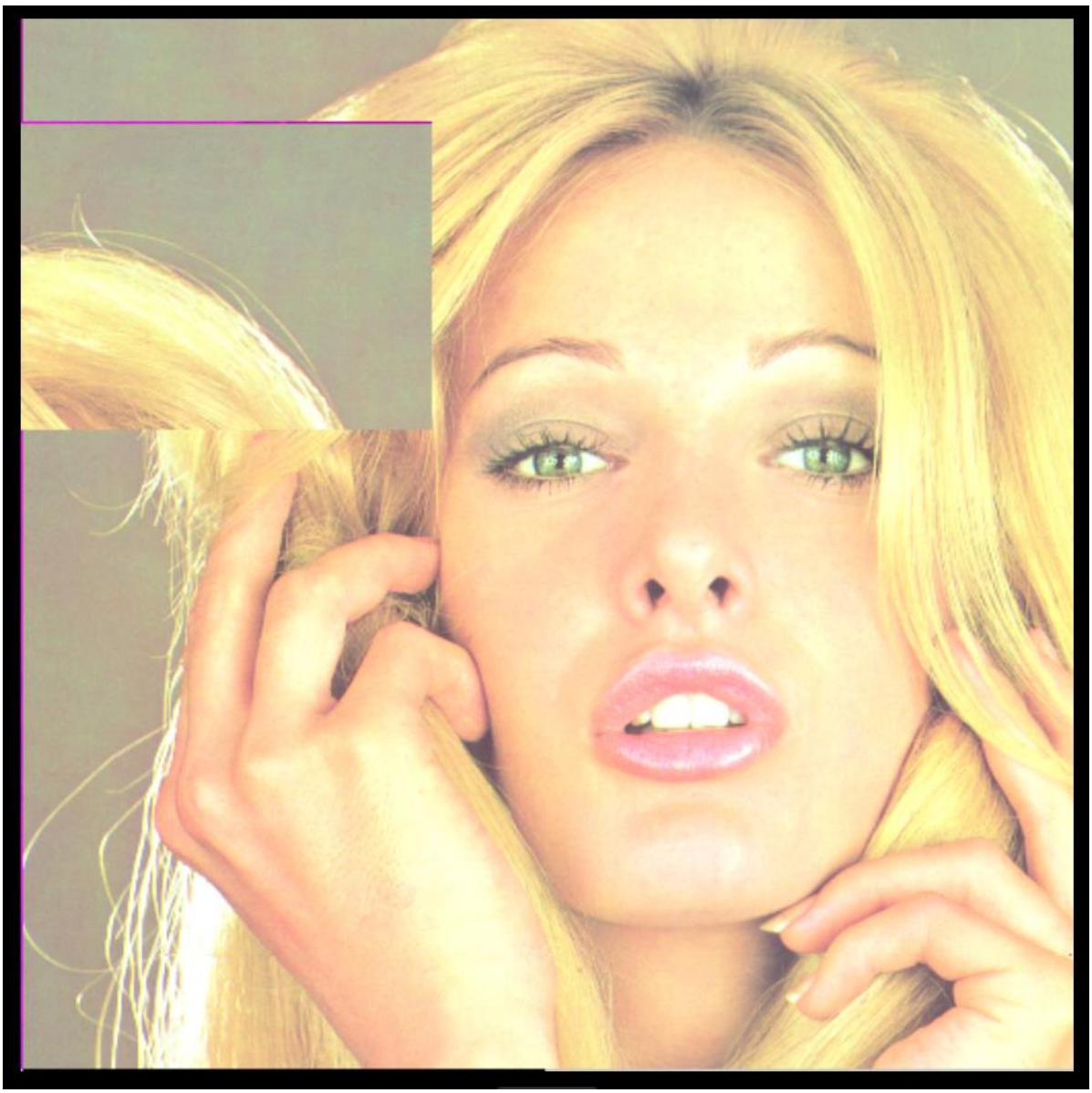
6. Функция поворота на 180

./cw --rotate --angle 180 --left_up 0.0 --right_down 550.550 /home/nikita/Загрузки/Tiffany.bmp

7. Функция поворота на 270

   ./cw --rotate --angle 270 --left_up 0.0 --right_down 150.250

   /home/nikita/Загрузки/Tiffany.bmp



8. Rgb-фильтр

   ./cw --rgbfilter --component_name red --component_value 56 --input /home/nikita/Загрузки/Lake.bmp

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.c

```
#include "BmpChange.h"
#include "BmpDraw.h"
#include "consol.h"
int main(int argc, char* argv[]) {
   char name[50];
   name[0]='\0';
   char file_outname[50] = "./out.bmp";
   const char* short_options = "Ho:I:i:";
   const struct option long_options[] = {
         { "help", no_argument, NULL, 'H' }, //
         { "output", required_argument, NULL, 'p' },
         { "info", required_argument, NULL, 'I' },//
         { "input", required_argument, NULL, 'i' },
         { "squared_lines", no_argument, NULL, 'S' }, //
         { "left_up", required_argument, NULL, 'u' },
         { "side_size", required_argument, NULL, 'l' }, //
         { "thickness", required_argument, NULL, 't' }, //
         { "color", required_argument, NULL, 'o' },
         { "fill", no_argument, NULL, 'b' }, //
         { "fill_color", required_argument, NULL, 'f' },

         { "rgbfilter", no_argument, NULL, 'F' }, //
         { "component_name", required_argument, NULL, 'n' }, //
         { "component_value", required_argument, NULL, 'v' }, //

         { "rotate", no_argument, NULL, 'R' }, //
         { "right_down", required_argument, NULL, 'd' },
         { "angle", required_argument, NULL, 'a' }, //

         { "circle", no_argument, NULL, 'C' }, //
         { "center", required_argument, NULL, 'c' },
         { "radius", required_argument, NULL, 'r' }, //
         { NULL, 0, NULL, 0 }
   };
   //printf("argc: %d\n", argc);
   char** arg_f1 = malloc(sizeof(char*)*6); //1.0 - left(), 1.1-side, 1.2 - thickness()
1.3 - color(), 1.4-flag(), 1.5 - color for fill()
   for (int i=0; i<6; i++){
```

```
        arg_f1[i] = malloc(sizeof(char)*10);
        arg_f1[i][0] = '\0';
    }
    char** arg_f2 = malloc(sizeof(char*)*2);
    for (int i=0; i<2; i++){
        arg_f2[i] = malloc(sizeof(char)*10);
        arg_f2[i][0] = '\0';     // заполняется полностью 2.0 - компонент, 2.1 -
значение от 0-255
    }
    char** arg_f3 = malloc(sizeof(char*)*3); // 3.0 - left(внешняя переменная), 3.1 -
right, 3.2 - угол
    for (int i=0; i<3; i++){
        arg_f3[i] = malloc(sizeof(char)*10);
        arg_f3[i][0] = '\0';
    }
    char** arg_f4 = malloc(sizeof(char*)*6); //4.0 - center, 4.1-radius, 4.2 - thickness()
4.3 - color, 4.4-flag, 4.5 - color for fill
    for (int i=0; i<6; i++){
        arg_f4[i] = malloc(sizeof(char)*10);
        arg_f4[i][0] = '\0';
    }
    // общие переменные
    int flag=0;
    int fill_flag;
    char thickness[15];
    thickness[0] = '\0';
    char left_up[20];
    left_up[0]='\0';
    char color[15];
    color[0] ='\0';
    char fill_color[15];
    fill_color[0] = '\0';
    //выделение под аргументы память
    int opt;
    opterr = 0;
    while ((opt = (getopt_long(argc, argv, short_options, long_options, NULL)))!=-1){
        switch(opt){
            case 'H':
                if (argc>2){
                    printf("Флаг help не ожидает дополнительных аргументов!\n");
                    exit(40);
                }
                flag = 7;
                break;
```

```
case 'S': //функции
case 'F':
case 'R':
case 'C':
case 'I':
    if (opt=='I'){
        if (argc<2){
            printf("Недостаточно параметров!\n");
            exit(40);
        }
        if (argc>3){
            printf("Флаг info не ожидает дополнительных параметров!\n");
            exit(40);
        }
    }
    else if(opt=='F' && argc <=6){
        printf("Недостаточно параметров!\n");
        exit(40);
    }
    else if(opt=='R' && argc <=8){
        printf("Недостаточно параметров!\n");
        exit(40);
    }
    else if((opt == 'C' || opt=='S') && argc<=10){
        printf("Недостаточно параметров!\n");
        exit(40);
    }
    if (opt=='S')flag =1;
    else if(opt=='F') flag=2;
    else if (opt=='R') flag=3;
    else if (opt=='C') flag=4;
    else flag=6;
    break;
case 'b': //флаг заливки
    fill_flag=1;//1.4, 4.4
    break;
case 'i':
case 'p':
    if (optarg[0] == '-'){
        printf("%s\n", "Аргумент пропущен!");
        exit(40);
    }
    if (opt=='p') strcpy(file_outname, optarg);
    if (opt=='i') strcpy(name, optarg);
```

```
    break;
case 'd':
case 'u': //координаты
case 'c':
    if (optarg[0] == '-' && optarg[1]=='-'){
        printf("%s\n", "Аргумент пропущен!");
        exit(40);
    }
    // error need
    char* ptr = malloc((strlen(optarg)+1)*sizeof(char));
    strcpy(ptr, optarg);
    char* token = strtok(ptr, ".");
    while (token!= NULL) {
        int j;
        if (token[0] == '-') j=1;
        else j=0;
        for (j;j<strlen(token);j++){
            if (!isdigit(token[j])){
                printf( "Аргумент не число!\n");
                exit(40);
            }
        }
        token = strtok(NULL, ".");
    }
    free(ptr);
    if (opt=='d'){
        strcpy(arg_f3[1], optarg); //3.1
    }
    else if(opt=='c'){
        strcpy(arg_f4[0], optarg); //4.0
    }
    else strcpy(left_up, optarg); //1.0, 3.0
    break;
case 'o': //цвета
case 'f':
    if (optarg[0] == '-' && optarg[1]=='-'){
        printf("Аргумент пропущен!\n");
        exit(40);
    }
    //error
    char* p = malloc((strlen(optarg)+1)*sizeof(char));
    strcpy(p, optarg);
    char* t = strtok(p, ".");
    while (t!= NULL) {
```

```
                if (strlen(t)>3){
                    printf( "Аргумент для компонента цвета не может быть больше 3х
значного числа и меньше нуля!\n");
                    exit(40);
                }
                for (int i = 0; i < strlen(t); i++) {
                    if (!isdigit(t[i])) {
                        printf( "Аргумент не число!\n");
                        exit(40);
                    }
                }
                if (atoi(t) > 255 || atoi(t) < 0) { //dop
                    printf("Аргумент вне диапазона RGB!\n");
                    exit(40);
                }
                t = strtok(NULL, ".");
            }
            free(p);
            if (opt == 'o') strcpy(color, optarg);
            else strcpy(fill_color, optarg); //1.5, 4.5
            break;
        case 'n':
            //printf("%s\n", optarg);
            if (optarg[0] == '-' ){
                printf( "Аргумент пропущен!\n");
                exit(40);
            }
            if (strcmp(optarg, "blue")!=0 && strcmp(optarg, "green")!=0 &&
strcmp(optarg, "red")!=0){
                printf( "Аргумент не цвет: 'blue', 'green', 'red'!\n");
                exit(40);
            }
            strcpy(arg_f2[0], optarg); //2.0
            break;
        case 'r':
        case 't': //радиус, толщина , цвет в 2й функции, длина стороны квадрата,
угол поворота изображения
        case 'v':
        case 'a':
        case 'l':
            //printf("%s\n", optarg);
            if (optarg[0] == '-' && optarg[1]=='-'){
                printf( "Аргумент пропущен!\n");
                exit(40);
```

```
            }
            int j;
            if (optarg[0] == '-') j=1;
            else j=0;
            for (j;j<strlen(optarg);j++){
               if (!isdigit(optarg[j])){
                  printf( "Аргумент не число!\n");
                  exit(40);
               }
            }
            if (opt=='v') {
               if (atoi(optarg) > 255 || atoi(optarg) < 0) {
                  printf("Аргумент вне диапазона RGB!\n");
                  exit(40);
               }
               strcpy(arg_f2[1], optarg); //2.1
            }
            if (opt != 'v' && atoi(optarg)<=0){
               printf("Аргумент должен быть больше 0!\n");
               exit(40);
            }
            if(opt=='r'){
               strcpy(arg_f4[1], optarg); //4.1
            }
            else if(opt=='a'){
               if (atoi(optarg)!=90 && atoi(optarg)!=270 && atoi(optarg)!=180){
                  printf( "Угол может быть только: 90, 180, 270!\n");
                  exit(40);
               }
               strcpy(arg_f3[2], optarg); //3.2
            }
            else if(opt=='l') strcpy(arg_f1[1], optarg); //1.1
            else{
               strcpy(thickness, optarg); //1.2, 4.2
            }
            break;
         default:
            printf("Несуществующий параметр или данный параметр не ожидался:
%s!\n", argv[optind-1]);
            exit(40);
      }
   };
   if (flag==7){
      help();
```

```
    for (int i=0; i<6; i++){
        free(arg_f1[i]);
        free(arg_f4[i]);
    }
    free(arg_f1);
    free(arg_f4);
    for (int i=0; i<2; i++){
        free(arg_f2[i]);
    }
    free(arg_f2);
    for (int i=0; i<3; i++){
        free(arg_f3[i]);
    }
    free(arg_f3);
    return 0;
}
//char name[50] = "/home/nikita/Загрузки/Lake.bmp";
if (strlen(name)==0) strcpy(name, argv[argc-1]);
BmpFileHeader* ls = (BmpFileHeader*)malloc(sizeof(BmpFileHeader));
BmpInfoHeader* rb = (BmpInfoHeader*)malloc(sizeof(BmpInfoHeader));
Rgb** our = read_bmp(name, ls, rb);
if(flag==6){
    info_printer(ls, rb);
    free_bmp(ls, rb, our);
    for (int i=0; i<6; i++){
        free(arg_f1[i]);
        free(arg_f4[i]);
    }
    free(arg_f1);
    free(arg_f4);
    for (int i=0; i<2; i++){
        free(arg_f2[i]);
    }
    free(arg_f2);
    for (int i=0; i<3; i++){
        free(arg_f3[i]);
    }
    free(arg_f3);
    return 0;
}
else if (flag==2){
    for (int i=0; i<2;i++){
        if (arg_f2[i][0]=='\0'){
            printf("Не все аргументы переданы для работы с флагом rgbfilter!\n");
```

```
            exit(40);
        }
    }
    int col=-1;
    if (strcmp(arg_f2[0], "blue")==0) col=0;
    else if (strcmp(arg_f2[0], "green")==0) col=1;
    else col=2;
    changer_color(our, col, atoi(arg_f2[1]), rb->width, rb->height);
}
else if(flag==3){
    if (strlen(left_up)!=0) strcpy(arg_f3[0], left_up);
    for (int i=0; i<3;i++){
        if (arg_f3[i][0]=='\0'){
            printf("Не все аргументы переданы для работы с флагом rotate!\n");
            exit(40);
        }
    }
    int x_l, y_l, x_r, y_r;
    sscanf(arg_f3[0], "%d.%d", &x_l, &y_l);
    sscanf(arg_f3[1], "%d.%d", &x_r, &y_r);
    rotate(our, atoi(arg_f3[2]), &(rb->height), &(rb->width), x_l, y_l, x_r, y_r);
}
else if(flag==1){
    if (strlen(left_up)!=0) strcpy(arg_f1[0], left_up);
    if (strlen(color)!=0) strcpy(arg_f1[3], color);
    if (strlen(thickness)!=0) strcpy(arg_f1[2], thickness);
    if (fill_flag==1){
        if (strlen(fill_color)!=0) strcpy(arg_f1[5], fill_color);
        else{
            printf("Не все аргументы переданы для работы с флагом
squared_lines!\n");
            exit(40);
        }
    }
    for (int i=0; i<4;i++){
        if (arg_f1[i][0]=='\0'){
            printf("Не все аргументы переданы для работы с флагом
squared_lines!\n");
            exit(40);
        }
    }
    int x_ll, y_ll;
    sscanf(arg_f1[0], "%d.%d", &x_ll, &y_ll);
```

```
        draw_square(our, x_ll, y_ll, arg_f1[3], atoi(arg_f1[1]), atoi(arg_f1[2]), fill_flag,
rb->height, rb->width, arg_f1[5]);
    }
    else if (flag==4){
        if (strlen(color)!=0) strcpy(arg_f4[3], color);
        if (strlen(thickness)!=0) strcpy(arg_f4[2], thickness);
        if (fill_flag==1){
            if (strlen(fill_color)!=0) strcpy(arg_f4[5], fill_color);
            else{
                printf("Не все аргументы переданы для работы с флагом circle!\n");
                exit(40);
            }
        }
        for (int i=0; i<4;i++){
            //printf("[%s]\n", arg_f4[i]);
            if (arg_f4[i][0]=='\0'){
                printf("Не все аргументы переданы для работы с флагом circle!\n");
                exit(40);
            }
        }
        draw_circle(our, arg_f4[0], atoi(arg_f4[1]), arg_f4[3], atoi(arg_f4[2]), fill_flag,
rb->height, rb->width, arg_f4[5]);
    }
    //
    FILE* new_bmp = fopen(file_outname, "wb");
    fwrite(ls, 1, sizeof(BmpFileHeader), new_bmp);
    fwrite(rb, 1, sizeof(BmpInfoHeader), new_bmp);
    for (int i = 0; i < rb->height; i++) {
        fwrite(our[i], 1, rb->width *sizeof(Rgb) + (4 - ((rb->width * 3) % 4)) % 4,
new_bmp);
    }
    fclose(new_bmp);
    for (int i=0; i<6; i++){
        free(arg_f1[i]);
        free(arg_f4[i]);
    }
    free(arg_f1);
    free(arg_f4);
    for (int i=0; i<2; i++){
        free(arg_f2[i]);
    }
    free(arg_f2);
    for (int i=0; i<3; i++){
        free(arg_f3[i]);
```

```
    }
    free(arg_f3);
    free_bmp(ls, rb, our);
}
```

Файл consol.h

```
#ifndef CONSOL_H
#define CONSOL_H
#include "BmpRead.h"
void help();
#endif //BMP_CW2_CONSOL_H
```

Файл consol.c

```
#include "consol.h"
void help(){
    printf("Course work for option 5.9, created by Nikita Matveev\n");
    printf("Программа предназначена для обработки BMP изображений: без
сжатия, 24 бита на пиксель.\n");
    printf("Доступный функционал:\n");
    printf("-h/--help  -  Вывод справки о программе\n");
    printf("-I/--info  -  Вывод информации об изображении\n");
    printf("-i/--input  -  Имя входного изображения\n");
    printf("-o/--output  -  Имя выходного изображения\n");
    printf("--squared_lines - Флаг для рисования квадрата с диагоналями. Квадрат
определяется:\n");
    printf("    --left_up  -  Координаты левого угла, значение задаётся в формате
`left.up`, где left – координата по x, up – координата по y\n");
    printf("    --side_size - Размер стороны, значение должно быть целым числом
больше 0\n");
    printf("    --thickness  -  Толщина линии, значение должно быть целым числом
больше 0\n");
    printf("    --color  -  Цвет линий, задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb
– числа, задающие цветовую компоненту\n");
    printf("    --fill - Наличие заливки, работает как бинарное значение: флага нет
– false , флаг есть – true.\n");
    printf("    --fill_color - Цвет заливки, задаётся строкой `rrr.ggg.bbb`, где
rrr/ggg/bbb – числа, задающие цветовую компоненту\n");
    printf("--rgbfilter - Флаг для изменения заданной компоненты цвета на
переданное значение. Функционал определяется:\n");
    printf("    --component_name - Имя компоненты, возможные значения `red`,
`green` и `blue`\n");
```

```
    printf("    --component_value - Принимает значение в виде числа от 0 до 255\n");
    printf("--rotate - Флаг для поворота изображения (части) на 90/180/270 градусов. Функционал определяется:\n");
    printf("    --left_up  -  Координаты левого угла, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y\n");
    printf("    --right_down  -  Координаты правого угла, значение задаётся в формате `right.down`, где right – координата по x, down – координата по y\n");
    printf("    --angle - Угол поворота, допустимые значения: `90`, `180`, `270`\n");
    printf("--circle - Флаг для рисования окружности. Окружность определяется:\n");
    printf("    --center - Координаты центра, задаётся в формате `x.y`, где x – координата по оси x, y – координата по оси y.\n");
    printf("    --radius - Радиус, значение должно быть целым числом больше 0\n");
    printf("    --thickness  -  Толщина линии, значение должно быть целым числом больше 0\n");
    printf("    --color  -  Цвет линий, задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту\n");
    printf("    --fill - Наличие заливки, работает как бинарное значение: флага нет – false , флаг есть – true.\n");
    printf("    --fill_color - Цвет заливки, задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту\n");
}
```

Файл BmpDraw.h

```
#ifndef BMPDRAW_H

#define BMPDRAW_H
#include "BmpRead.h"
void draw_square(Rgb** pixels, int x_l, int y_l, char color[], int side_size, int thickness, int flag, int H, int W, char c_fill[]);
void draw_circle(Rgb** pixels, char coord[], int radius, char color[], int thickness, int flag, int H, int W, char c_fill[]);
#endif //BMP_CW2_BMPDRAW_H
```

Файл BmpDraw.c

```
#include "BmpDraw.h"
void draw_square(Rgb** pixels, int x_l, int y_l, char color[], int side_long, int thickness, int flag, int H, int W, char c_fill[]){
    int b, g, r;
    sscanf(color, "%d.%d.%d", &r, &g, &b);
    //printf("%d %d %d", b, g, r);
    //цвета
    //printf("%d", thickness);
```

```
    int t_down, t_up;
    if (thickness % 2 !=0){
        t_down =  thickness/2;
        t_up = thickness/2;
    }
    else{
        t_down =thickness/2;
        t_up = thickness/2 -1;
    }
    if (x_l > W && y_l>H) return;
    //толщина для квадрата
    int r_wall=0;
    int d_wall=0;
    int l_wall=0;
    int up_wall=0;
    if (x_l+side_long>W) r_wall = W;
    if (x_l <0){
        if (x_l+side_long < 0) return;
        l_wall = -1;
    }
    if (y_l<0){
        if (H-1-y_l-side_long>H) return;
        up_wall = H;
    }
    if (y_l>=H){
        return;
    }
    if (x_l>=W) return;
    y_l = H-1-y_l;
    if (y_l-side_long<0) d_wall = -1;
    //if (((l_wall == -1) & (r_wall == W)) || ((up_wall == H) & (d_wall == -1))) return;
// без диагоналей пока что
    // границы
    if (flag==1){
        int b_f, g_f, r_f;
        sscanf(c_fill, "%d.%d.%d", &r_f, &g_f, &b_f);
        int x1 =x_l+1;
        int y1 =y_l-1;
        int x2 = x1+side_long-1;
        int y2= y_l-side_long+1;
        if (up_wall == H) y1 = H-1;
        if (l_wall==-1) x1= 0;
        if (d_wall ==-1) y2=0;
        if (r_wall==W) x2= W-1;
```

```
    for (int y=y2; y<=y1; y++){
        for (int x=x1; x<=x2; x++){
            pixels[y][x].b=b_f;
            pixels[y][x].g=g_f;
            pixels[y][x].r=r_f;
        }
    }
}
//заливка
for (int c=-t_down; c<t_up+1; c++){
    for (int x=x_l+c; x<=x_l+side_long-c; x++){
        if (x<0) continue;
        if (x==W) break;
        if (up_wall!=H){
            if (y_l-c>=0 && y_l-c<=H-1){ //+c was
                pixels[y_l-c][x].b=b; //+c вроде вниз
                pixels[y_l-c][x].g=g;
                pixels[y_l-c][x].r=r;
            }
        }
        if (d_wall != -1){
            if (y_l-side_long+c>=0 && y_l-side_long+c<=H-1){
                pixels[y_l-side_long+c][x].b=b;
                pixels[y_l-side_long+c][x].g=g;
                pixels[y_l-side_long+c][x].r=r;
            }
        }
    }
}
// верх и низ
for (int c=-t_down; c<t_up+1; c++){
    for (int y=y_l-side_long-t_down; y<=y_l+t_up; y++){
        if (y<0) continue;
        if (y==H) break;
        if (l_wall !=-1){
            if (x_l+c >=0 && x_l+c<W){
                pixels[y][x_l+c].b=b;
                pixels[y][x_l+c].g=g;
                pixels[y][x_l+c].r=r;
            }
        }
        if (r_wall!=W){
            if (x_l+side_long+c >=0 && x_l+side_long+c<W) {
                pixels[y][x_l + side_long - c].b = b;
```

```
                pixels[y][x_l + side_long - c].g = g;
                pixels[y][x_l + side_long - c].r = r; //+c внутрь будет при четных
            }
        }
    }
}
// бока
int x_c1, x_c2, y_c1, y_c2;  //1 ая снизу слева, вторая сверху справа
x_c1 = x_l+1;
x_c2 = x_l+side_long-1;
y_c1 = y_l-side_long;
y_c2 = y_l-1;
//printf("[%d]x_c1:%d y_c1:%d %d %d", x_l, x_c1, y_c1, x_c2, y_c2);
//координаты для диагонали 1
int dx = x_c2-x_c1;
int dy = y_c2-y_c1;
int di_1 = 2*dy-dx;
int d1 = 2*dy;
int d2 = 2*(dy-dx);
for (int c=-t_down; c<t_up+1; c++){
    int yy=y_c1;
    for (int x=x_c1; x<=x_c2; x++){
        if (di_1<=0) di_1+=d1;
        else{
            yy++;
            di_1+=d2;
        }
        if (x==W || yy+c==H) break;
        if (x>0 && yy+c<=H && yy+c>=0){
            pixels[yy+c][x].b = b;
            pixels[yy+c][x].g = g;
            pixels[yy+c][x].r = r;
        }
    }
}
//1ая диагональ
int x_c3, x_c4, y_c3, y_c4; // 3ая вверху слева
x_c3 = x_l+1;
x_c4 = x_l+side_long-1;
y_c3 = y_l;
y_c4 = y_l-side_long;
int dx_2 = x_c4-x_c3;
int dy_2 = y_c3 - y_c4;
int di_2 = 2*dy_2 - dx_2;
```

```
    int d1_2 =2*dy_2;
    int d2_2 =2*(dy_2-dx_2);
    for (int c=-t_down; c<t_up+1; c++){
        int yyy=y_c3;
        for (int x=x_c3; x<=x_c4; x++){
            if (di_2<=0) di_2+=d1_2;
            else{
                yyy--;
                di_2+=d2_2;
            }
            if (x==W) break; //was yyy<0
            if (x>=0 && yyy+c>=0 && yyy+c<H){
                pixels[yyy+c][x].b = b;
                pixels[yyy+c][x].g = g;
                pixels[yyy+c][x].r = r;
            }
        }
    }
    // 2ая диагональ
}

void draw_circle(Rgb** pixels, char coord[], int radius, char color[], int thickness, int
flag, int H, int W, char c_fill[]){
    int b, g, r;
    sscanf(color, "%d.%d.%d", &r, &g, &b);
    //цвета
    int x, y;
    sscanf(coord, "%d.%d", &x, &y);
    y=H-1-y;
    //координаты
    /*
    pixels[y][x].b=0;
    pixels[y][x].g=0;
    pixels[y][x].r=0;*/
    //центр закрашивать
    int t_down, t_up;
    if (thickness % 2 !=0){
        t_down =  thickness/2;
        t_up = thickness/2;
    }
    else{
        t_down =thickness/2;
        t_up = thickness/2 -1;
    }
```

```
//printf("%d %d", t_down, t_up);
//толщина

if (flag==1){
    int b_f, g_f, r_f;
    sscanf(c_fill, "%d.%d.%d", &r_f, &g_f, &b_f);
    int x_l, y_l, x_r, y_r;
    x_l = x-radius;
    x_r = x+radius;
    y_l = y+radius;
    y_r = y-radius;
    for (int i=y_r; i<y_l; i++){
        if (i < 0 || i>=H) continue;
        for (int j=x_l; j<x_r; j++){
            if (j>=W || j<0) continue;
            if ((x-j)*(x-j) + (y-i)*(y-i) < radius*radius){
                pixels[i][j].b=b_f;
                pixels[i][j].g=g_f;
                pixels[i][j].r=r_f;
            }
        }
    }
} // заливка
for (int k=-t_down; k<t_up+1; k++){
    int xi = 0;
    int rr = radius+k;
    int yi=rr;
    int dd = 2*(xi-yi+1);
    if (y+yi < H && y+yi>=0 && x+xi<W && x+xi>=0){
            pixels[y+yi][x+xi].b=b;
            pixels[y+yi][x+xi].g=g;
            pixels[y+yi][x+xi].r=r;
        }
    if (y-yi>=0 && y-yi<H && x+xi>=0 && x+xi <W){
            pixels[y-yi][x+xi].b=b;
            pixels[y-yi][x+xi].g=g;
            pixels[y-yi][x+xi].r=r;
        }
    // центральные пиксели под и над радиусом
    while (yi>0){
        int di, si;
        if (dd==0){
            xi++;
            yi--;
```

```
    dd+=2*(xi-yi+1);

}
else if(dd<0){
    di = 2*(dd+yi)-1;
    if (di<=0){
        xi++;
        dd+=2*xi+1;

    }
    else{
        xi++;
        yi--;
        dd+=2*(xi-yi+1);

    }
}
else{
    si = 2*(dd-xi)-1;
    if (si<=0){
        xi++;
        yi--;
        dd+=2*(xi-yi+1);

    }
    else{
        yi--;
        dd+=1-2*yi;

    }
}
if (y+yi <H && y+yi>=0) {
    //правая сторона
    if (x + xi < W && x + xi >= 0) {
        pixels[y + yi][x + xi].b = b;
        pixels[y + yi][x + xi].g = g;
        pixels[y + yi][x + xi].r = r; //основная окружность толщины 1
    }
    if (x - xi >= 0 && x - xi < W) { //левая сторона
        pixels[y + yi][x - xi].b = b;
        pixels[y + yi][x - xi].g = g;
        pixels[y + yi][x - xi].r = r;
    }
}
```

```
            //верхняя часть
            if (y-yi >=0 && y-yi<H){
                if (x+xi<W && x+xi>=0){
                    pixels[y-yi][x+xi].b=b;
                    pixels[y-yi][x+xi].g=g;
                    pixels[y-yi][x+xi].r=r;

                }
                if (x-xi>=0 && x-xi<W){
                    pixels[y-yi][x-xi].b=b;
                    pixels[y-yi][x-xi].g=g;
                    pixels[y-yi][x-xi].r=r;
                }
            }
            //нижняя часть
        }
        // круг
    }
    int x_ll, x_rr, y_ll, y_rr;
    int r_max = radius + t_up;
    int r_min = radius - t_down;
    if (r_min<0){
        r_min=0;
    }
    x_rr = x+r_max;
    x_ll = x-r_max;
    y_rr = y-r_max;
    y_ll = y+r_max;
    if (x_rr<=W) x_rr=W-1;
    if (x_ll<0) x_ll=0;
    if (y_rr<0) y_rr=0;
    if (y_ll>=H) y_ll=H-1;
    for (int i=y_rr; i<y_ll; i++) {
        if (i < 0 || i >= H) continue;
        for (int j = x_ll; j < x_rr; j++) {
            if (j >= W || j < 0) continue;
            if ((x-j)*(x-j) + (y-i)*(y-i) >= r_min*r_min && (x-j)*(x-j) + (y-i)*(y-i) <=
r_max*r_max){
                pixels[i][j].b=b;
                pixels[i][j].g=g;
                pixels[i][j].r=r;
            }
        }
    }
```

```
    //толщина
}
```

Файл BmpRead.h

```
#ifndef BMPREAD_H
#define BMPREAD_H
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <locale.h>
#include <unistd.h>
#include <getopt.h>
#include <ctype.h>
#pragma pack (1)
typedef struct {
    unsigned char signature[2];
    unsigned int filesize;
    unsigned short reserved1;
    unsigned short reserved2;
    unsigned int pixelArrOffset;
} BmpFileHeader;

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

typedef struct {
    unsigned char b;
    unsigned char g;
    unsigned char r;
} Rgb;
#pragma pop
Rgb** read_bmp(char name[], BmpFileHeader* b1, BmpInfoHeader* b2);
void free_bmp(BmpFileHeader* b1, BmpInfoHeader* b2, Rgb** arr);
```

```
void info_printer(BmpFileHeader* b1, BmpInfoHeader* b2);
#endif //BMPREAD_H"
```

Файл BmpRead.c

```
#include "BmpRead.h"

Rgb** read_bmp(char file_name[], BmpFileHeader* bmfh, BmpInfoHeader* bmif){
   FILE *f = fopen(file_name, "r");
   if (!f){
      printf("Невозможно открыть файл!\n");
      exit(40);
   }
   fread(bmfh, 1, sizeof(BmpFileHeader), f);
   if (strncmp(bmfh->signature, "BM", 2)!=0){
      printf("Файл не формата Bmp!\n");
      exit(40);
   }
   fread(bmif, 1, sizeof(BmpInfoHeader), f);

   if (bmif->compression !=0){
      printf("Файл сжат!\n");
      exit(40);
   }
   if (bmif->bitsPerPixel !=24){
      printf("Недопустимая глубина цвета!\n");
      exit(40);
   }
   unsigned int H = bmif->height;
   unsigned int W = bmif->width;
   Rgb** arr = malloc(H * sizeof(Rgb*)); //строки пикселей по факту
   for(int i = 0; i < H; i++){
      arr[i] = malloc(W * sizeof(Rgb) + (4 - ((W * 3) % 4)) % 4); //под 1 строку
выделяем
      fread(arr[i], 1, W * sizeof(Rgb) + (4 - ((W * 3) % 4)) % 4,f);
   }
   fclose(f);
   return arr;
}

void free_bmp(BmpFileHeader* bmfh, BmpInfoHeader* bmif, Rgb** pixels){
   unsigned int H = bmif->height;
   free(bmfh);
```

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

void info_printer(BmpFileHeader* bmfh, BmpInfoHeader* bmif){
    printf("Signature: %c%c\n", bmfh->signature[0], bmfh->signature[1]);
    printf("FileSize: %d\n", bmfh->filesize);
    printf("PixelArrOffset: %d\n", bmfh->pixelArrOffset);
    printf("HeaderSize: %d\n", bmif->headerSize);
    printf("Width: %d\n", bmif->width);
    printf("Height: %d\n", bmif->height);
    printf("Planes: %d\n", bmif->planes);
    printf("BitsPerPixel: %d\n", bmif->bitsPerPixel);
    printf("Compression: %d\n", bmif->compression);
    printf("ImageSize: %d\n", bmif->imageSize);
    printf("xPixelsPerMeter: %d\n", bmif->xPixelsPerMeter);
    printf("yPixelsPerMeter: %d\n", bmif->yPixelsPerMeter);
    printf("ColorsInColorTable: %d\n", bmif->colorsInColorTable);
    printf("ImportantColorCount: %d\n", bmif->importantColorCount);
}
```

Файл BmpChange.h

```
#ifndef BMPCHANGE_H
#define BMPCHANGE_H
#include "BmpRead.h"
void changer_color(Rgb** pixels, int col, int new_fill, int W, int H);
void rotate(Rgb** pixels, int degree, int* H, int* W, int x_l, int y_l, int x_r, int
y_r);
#endif //BMP_CW2_BMPCHANGE_H
```

Файл BmpChange.c

```
#include "BmpChange.h"
void changer_color(Rgb** pixels, int col, int new_fill, int W, int H){
    for (int i=0; i<H;i++){
        for (int j=0; j<W;j++){
            if (col==0) pixels[i][j].b = new_fill;
            else if(col==1) pixels[i][j].g = new_fill;
            else if (col==2) pixels[i][j].r = new_fill;
        }
    }
```

```
}

void rotate(Rgb** pixels, int degree, int* Hi, int* Wi, int x_l, int y_l, int x_r, int y_r){
   int H=*Hi;
   int W=*Wi;
   if (x_l==x_r && y_l==y_r){
      return;
   }
   //
   if (x_l>x_r){
      int x_pr = x_l;
      x_l=x_r;
      x_r=x_pr;
   }
   if (y_l>y_r){
      int y_pr = y_l;
      y_l=y_r;
      y_r=y_pr;
   }
   if(y_l>=(*Hi) || x_l>=(*Wi)){
      return;
   }
   // неправильные входные координаты

   if (x_l<0 && y_l <0){
      x_l =0;
      y_l=0;
   }
   else if(y_l<0) y_l=0;
   else if(x_l<0) x_l=0;

   if(x_r>=*Wi && y_r>=*Hi){
      x_r = (*Wi)-1;
      y_r= (*Hi)-1;
   }
   else if(y_r>=*Hi) y_r=(*Hi)-1;
   else if(x_r>=*Wi) x_r=(*Wi)-1;
   //максимальные координаты
   int flag = 0;
   if ((degree==90 || degree==270) && (y_l<=0 && x_l<=0 && x_r >= (*Wi)-1 && y_r >= (*Hi)-1)) flag=1;
  // if (y_l<=0 && x_l<=0 && x_r >= (*Wi)-1 && y_r >= (*Hi)-1) flag=1;
   //flag
```

```
//printf("H:%d\n", H);
//printf("y:%d\n", y_r);
y_l = H-y_l-1;
y_r = H-y_r-1;
H=abs(y_l-y_r)+1;
W=abs(x_r-x_l)+1;
//printf("H:%d\n", H);
//printf("W:%d\n", W);
//printf("good - x_r, y_r:[%d, %d]; x_l, y_l:[%d, %d]\n", x_r, y_r, x_l, y_l);
//координаты смена
Rgb** arr = malloc(H * sizeof(Rgb*));
int p_i = y_r;
//if (p_i != 0) p_i++;
for(int i = 0; i < H; i++) {
    arr[i] = malloc(W * sizeof(Rgb));
    int p_j= x_l;
    //if (p_i>=(*Hi)) break;
    for (int j = 0; j < W; j++) {
        arr[i][j] = pixels[p_i][p_j];
        p_j++;
    }
    p_i++;
}//копия для пикселей указанной области из преданного
if (degree==180){
    int c=1;
    if (flag==2){
        c =0;
        for (int y = y_l; y>=y_r; y--){
            int k=W-1;
            for (int x = x_l; x<=x_r; x++){
                pixels[y][x] = arr[c][k--];
            }
            c++;
        }
    }
    else{
        for (int y = y_l; y>y_r; y--){
            int k=W-2;
            for (int x = x_l; x<x_r; x++){
                pixels[y][x] = arr[c][k--];
            }
            c++;
        }
    }
```

```
    }
    else if(degree==90 || degree==270){
        int x_ll, x_rr, y_ll, y_rr;
        int c;
        if (flag==1){ //full picture
            for (int i=0; i<H;i++){
                free(pixels[i]);
            }
            pixels = (Rgb**)realloc(pixels, W * sizeof(Rgb*));
            for (int i=0; i<W;i++){
                pixels[i] = malloc(H * sizeof(Rgb) + (4 - ((H * 3) % 4)) % 4); //меняем
размер массива пикселей
            }
            (*Wi)=H; //меняем исходные W & H
            (*Hi)=W;
            x_ll=0;
            x_rr=(*Wi);
            y_ll=(*Hi)-1;
            y_rr=-1;
            if (degree==270){
                c = 0;
                for (int x=x_ll; x<x_rr; x++){
                    int k=0;
                    for (int y=y_ll; y>y_rr;y--){
                        pixels[y][x]=arr[c][k++];
                    }
                    c++;
                }
            }
            else{ //90 degree
                c = H-1;
                for (int x=x_ll; x<x_rr; x++){
                    int k=W-1;
                    for (int y=y_ll; y>y_rr;y--){
                        pixels[y][x]=arr[c][k--];
                    }
                    c--;
                }
            }
        }

        else{ // part
            if (degree==270){ //заполняем слева направоо сверху вниз
                c=1;
```

```
        x_ll = (x_l+x_r)/2 - (H-1)/2;
        y_ll = (y_l +y_r+1)/2 + (W-1)/2;
        x_rr = (x_ll+(y_l-y_r));
        y_rr = (y_ll-(x_r-x_l));
        for (int x=x_ll; x<x_rr; x++){
            if (x<0 || x>=(*Wi)){
                c++;
            }
            else{
                int k=0;
                for (int y=y_ll; y>y_rr;y--){
                    if (y<(*Hi) && y>=0){
                        pixels[y][x]=arr[c][k++];
                    }
                    else k++;
                }
                c++;
            }
        }
    }
    else{//заполняем слева направо сверху вниз
        c = H-1;
        x_ll = (x_l+x_r)/2 - (H-1)/2;
        y_ll = (y_l+y_r+1)/2 + (W-1)/2;
        x_rr = (x_ll+(y_l-y_r));
        y_rr = (y_ll-(x_r-x_l));
        for (int x=x_ll; x<x_rr; x++){
            if (x<0 || x>=(*Wi)) {
                c--;
            }
            else{
                int k=W-2;
                //printf("%d ", x);
                for (int y=y_ll; y>y_rr;y--){
                    if (y<(*Hi) && y>=0){
                        pixels[y][x]=arr[c][k--];
                    }
                    else k--;
                }
                c--;
            }
        }
    }
}
```

```
    }//90 and 270
    for (int i=0; i<H; i++){
        free(arr[i]);
    }
    free(arr);
}
```