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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка изображений

Студент гр. 3344 _____ **Сьомак Д.А.**

Преподаватель _____ **Глазунов С.А.**

Санкт-Петербург
2024

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Сьомак Д.А.

Группа 3344

Тема работы: Обработка изображений.

Исходные данные:

- Программа **обязательно должна иметь CLI** (опционально дополнительное использование GUI).
- Программа должна реализовывать весь следующий функционал по обработке bmp-файла
- 24 бита на цвет
- без сжатия
- файл может не соответствовать формату BMP, т.е. необходимо проверка на BMP формат (дополнительно стоит помнить, что версий у формата несколько). Если файл не соответствует формату BMP или его версии, то программа должна завершиться с соответствующей ошибкой.
- обратите внимание на выравнивание; мусорные данные, если их необходимо дописать в файл для выравнивания, должны быть нулями.
- обратите внимание на порядок записи пикселей
- все поля стандартных BMP заголовков в выходном файле должны иметь те же значения что и во входном (разумеется, кроме тех, которые должны быть изменены).
- Каждую подзадачу следует вынести в отдельную функцию, функции сгруппировать в несколько файлов
- Сборка должна осуществляться при помощи make и Makefile или другой системы сборки

Содержание пояснительной записки:

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

Предполагаемый объем пояснительной записки:
Не менее 30 страниц.

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 22.05.2024

Дата защиты реферата: 22.05.2024

Студент		Сьомак Д.А.
Преподаватель		Глазунов С.А.

АННОТАЦИЯ

Курсовая работа подразумевает написание программы, которая обрабатывает bmp-файл с использованием CLI интерфейса. Программа производит считывание и обработку текстовых файлов по заданным в командной строке флагам и параметрам. При написании программы использовались методы работы с изображением, структурами, динамической памятью и функциями стандартной библиотеки. Обработка bmp-файлов включает в себя 4 функции обработки изображения (инверсированные цвета, преобразование в Ч/Б, изменение размеров изображения, рисование отрезка). Результатом работы программы является обработанное изображение, которое будет сохранено в файл с заданным именем. Также результатом работы программы может быть справка о реализованных внутри программы функциях, полная информация о считанном bmp-файле или же ошибка с указанием на её причину.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной работы является создание программы на языке программирования C, которая будет обрабатывать BMP-изображение с помощью CLI интерфейса.

Для достижения поставленной цели требуется решить следующие задачи:

1. Изучить формат BMP
2. Изучить методы реализации CLI интерфейса
3. Реализовать функций обработки изображения
4. Реализовать эффективную сборку программы
5. Предусмотреть возможные ошибки и их причины

Возможные методы решения поставленных задач:

1. Реализация функций считывания и записи bmp-файлов
2. Реализация структур-хедеров для считанного изображения
3. Использование библиотеки getopt для работы с командной строкой
4. Сборка проекта с помощью Makefile
5. Вынесение каждой подзадачи в отдельную функцию

1. ОПИСАНИЕ ЗАДАНИЯ

Программа должна иметь следующую функции по обработке изображений:

Инверсия цвета в заданной области. Флаг для выполнения данной операции: `--inverse`. 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

Преобразовать в Ч/Б изображение (формулу можно посмотреть на wikipedia). Флаг для выполнения данной операции: `--gray`. 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

Изменение размера изображения с его обрезкой или расширением фона. Флаг для выполнения данной операции: `--resize`. Функционал определяется:

Количеством изменения пикселей с определенной стороны в формате: `--<side> <change>`, где `<side>` может принимать значения left (с левой стороны изменение), right (с правой стороны), above (с верхней стороны), below (с нижней стороны); `<side>` является числом: положительное означает расширение, отрицательное означает обрезку. Например, следующие флаги `--resize --left 100 --above -100 --below 30 --right -20` означает, что нужно расширить изображение слева на 100 пикселей и снизу на 30, и обрезать изображение сверху на 100 пикселей и справа на 20 пикселей.

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

Рисование отрезка. Флаг для выполнения данной операции: `--line`. Отрезок определяется:

координатами начала. Флаг `--start`, значение задаётся в формате `x.y`, где x – координата по x, y – координата по y

координатами конца. Флаг `--end` (аналогично флагу `--start`)

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

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

2. ОПИСАНИЕ ПРОГРАММЫ

2.1. Реализованные функции, структуры

Во время разработки программы были реализованы структуры:

1. Rgb - используется для хранения цвета пикселя.

2. BitmapFileHeader - используется для хранения общей информации об изображении.

3. BitmapInfoHeader - используется для хранения подробной информации об изображении и определения формата пикселей.

4. Image - используется для хранения массива пикселей и хедеров изображения.

5. Coords - используется для хранения координат конкретной точки.

6. image_resize - используется для хранения данных, необходимых для запуска функции resize.

7. line - используется для хранения данных, необходимых для запуска функции line.

8. flags - используется для хранения статуса флагов, поданных в командную строку.

9. struct option long_options[] - структура библиотеки getopt.

Во время разработки программы были реализованы функции:

1. void print_imageinfo(BitmapInfoHeader bmih, BitmapFileHeader bmfh) - функция, которая выводит подробную информацию о считанном изображении.

2. void read_bmp(char file_name[], Image *img) - функция считывания bmp-файла.

3. void write_bmp(char file_name[], Image img) - функция записи bmp-файла.

4. void coordschecker(char * cords) - функция, проверяющая координаты на корректность.

5. void distancechecker(char * distance) - функция, проверяющая дистанцию на корректность.

6. int *parse_color(char *color) - функция, которая делит цвета, поданные через точку на отдельные числа.

7. void colorchecker(char *color) - функция, проверяющая цвет на корректность.

8. void thickchecker(char *thickness) - функция, проверяющая толщину на корректность.

9. void nameschecker(char* inputname, char* outputname) - функция, проверяющая совпадение названий входного и выходного файлов.

10. void count_argscheck(char * arg1, char * arg2, char * arg3, char * name) - функция, проверяющая количество аргументов у флага, который их требует.

11. void no_argschecker(char* arg1, char* arg2, char *name) - функция, проверяющая количество аргументов у флага, который их принимает.

12. void description(void) - функция, которая выводит справку о реализованных в программе функциях.

13. void drawpixel(Rgb **arr, int W, int H,int y, int x, Rgb color) - функция, которая рисует пиксель заданного цвета.

14. void inverse(Image * img,int lu_x, int lu_y, int rd_x, int rd_y) - функция, которая инверсирует цвет в конкретной области изображения.

15. void black_white(Image *img,int lu_x, int lu_y, int rd_x, int rd_y) - функция, которая преобразует в Ч/Б конкретную область изображения.

16. void resize(Image * img, char side[], int distance,Rgb color) - функция, которая расширяет или обрезает изображение.

17. void fill_circle(Image *img, int x0, int y0, int rad,Rgb color) - функция, которая рисует залитый круг с нужным радиусом.

18. void drawThickLine(Image *img, int x1, int y1, int x2, int y2, int thickness, Rgb color) - функция, рисующая отрезок с конкретной толщиной на изображении.

19. void swap_int(int *a, int *b) - функция, меняющая местами значения переданных переменных.

20. unsigned int padding(unsigned int w) - функция, которая находит необходимое строке пикселей выравнивание.

21. unsigned int row_len(unsigned int w) - функция, которая находит длину строки пикселей с учётом выравнивания.

22. int main(int argc,char** argv) - основная функция программы, которая считывает опции командной строки, выполняет действия, которые им соответствует, и завершает работу.

2.2. Файловая структура программы

Во время разработки программа была разбита на следующие файлы:

- Makefile - файл, необходимый для компиляции и сборки проекта.
- Bmpfunc.c - файл, содержащий функции считывания, записи, вывода информации о bmp-файле.
- Bmpfunc.h - заголовочный файл, содержащий прототипы функций считывания, записи, вывода информации о bmp-файле.
- checkers.c - файл, содержащий функции, проверяющие данные на корректность.
- checkers.h - заголовочный файл, содержащий прототипы функций, проверяющих данные.
- cli.c - файл, содержащий реализацию CLI интерфейса.
- functions.c - файл, содержащий функции обработки изображения.
- functions.h - заголовочный файл, содержащий прототипы функций, обрабатывающих изображение.
- secondary_func.c - файл, содержащий вторичные функции, необходимые при работе других функций.
- secondary_func.h - заголовочный файл, содержащий прототипы вторичных функций.
- structurs.h - заголовочный файл, содержащий структуры и библиотеки.

2.3. Модульная структура, сборка

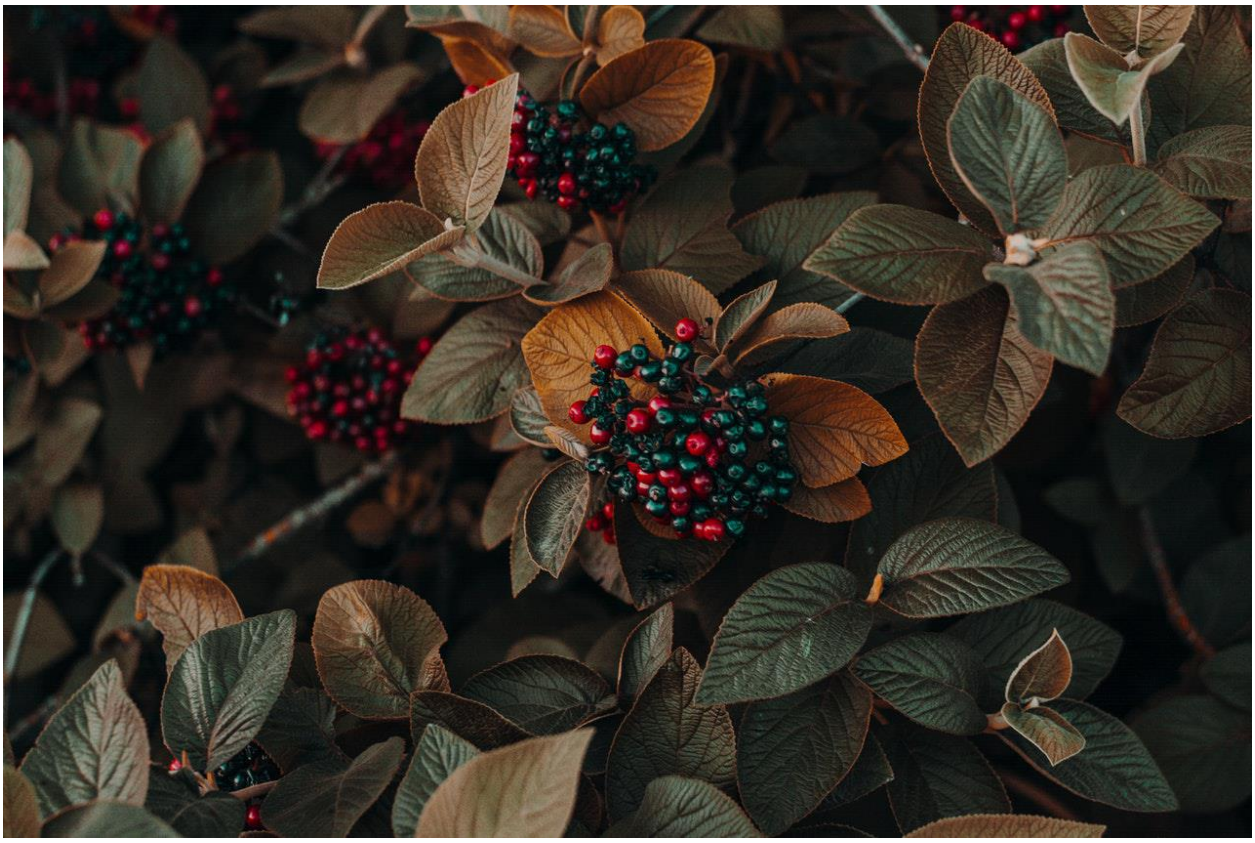
Для сборки проекта используется Makefile:

- cw – исполняемый файл, требует все нижеперечисленные объектные файлы для сборки и линковки.
- checkers.o - объектный файл, требующий checkers.c, structurs.h, checkers.h для компиляции.
- bmpfunc.o - объектный файл, требующий bmpfunc.c, structurs.h, bmpfunc.h, secondary_func.h для компиляции.
- secondary_func.o - объектный файл, требующий secondary_func.c, structurs.h, secondary_func.h для компиляции.
- functions.o - объектный файл, требующий functions.c, structurs.h, functions.h, secondary_func.h для компиляции.
- cli.o - объектный файл, требующий structurs.h, checkers.h, bmpfunc.h, checkers.h для компиляции.
- Clean - очистка всех объектных файлов и исполняемого файла cw.

Компиляция происходит с помощью: gcc.

3. ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

Исходная картинка(ex.bmp):

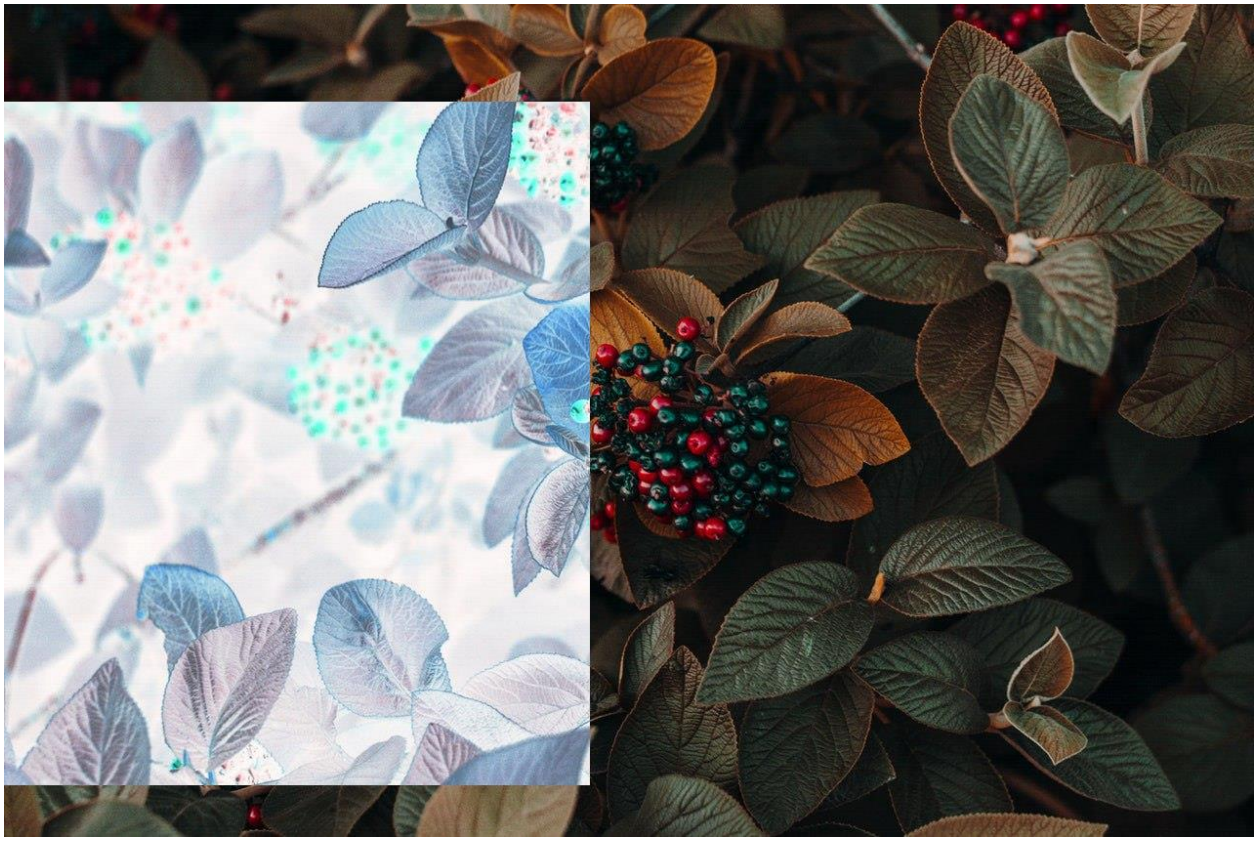


1. Инверсирование области изображения:

   Входные данные:

--right_down 600.800 --left_up -100.100 --inverse ex.bmp

   Обработанное изображение:

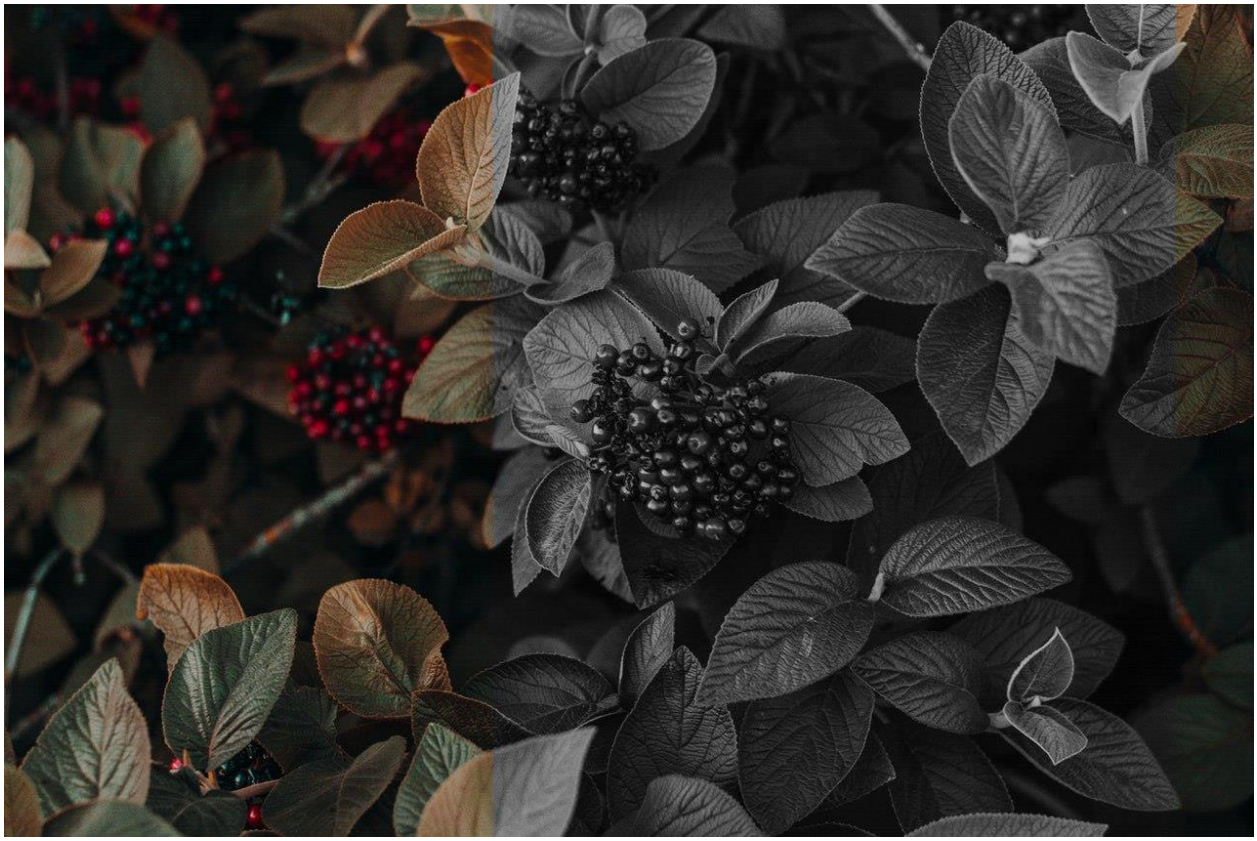


2. Преобразование области изображения в Ч/Б:

Входные данные:

--gray --right_down 1200.2000 --left_up 500.0 ex.bmp

Обработанное изображение:

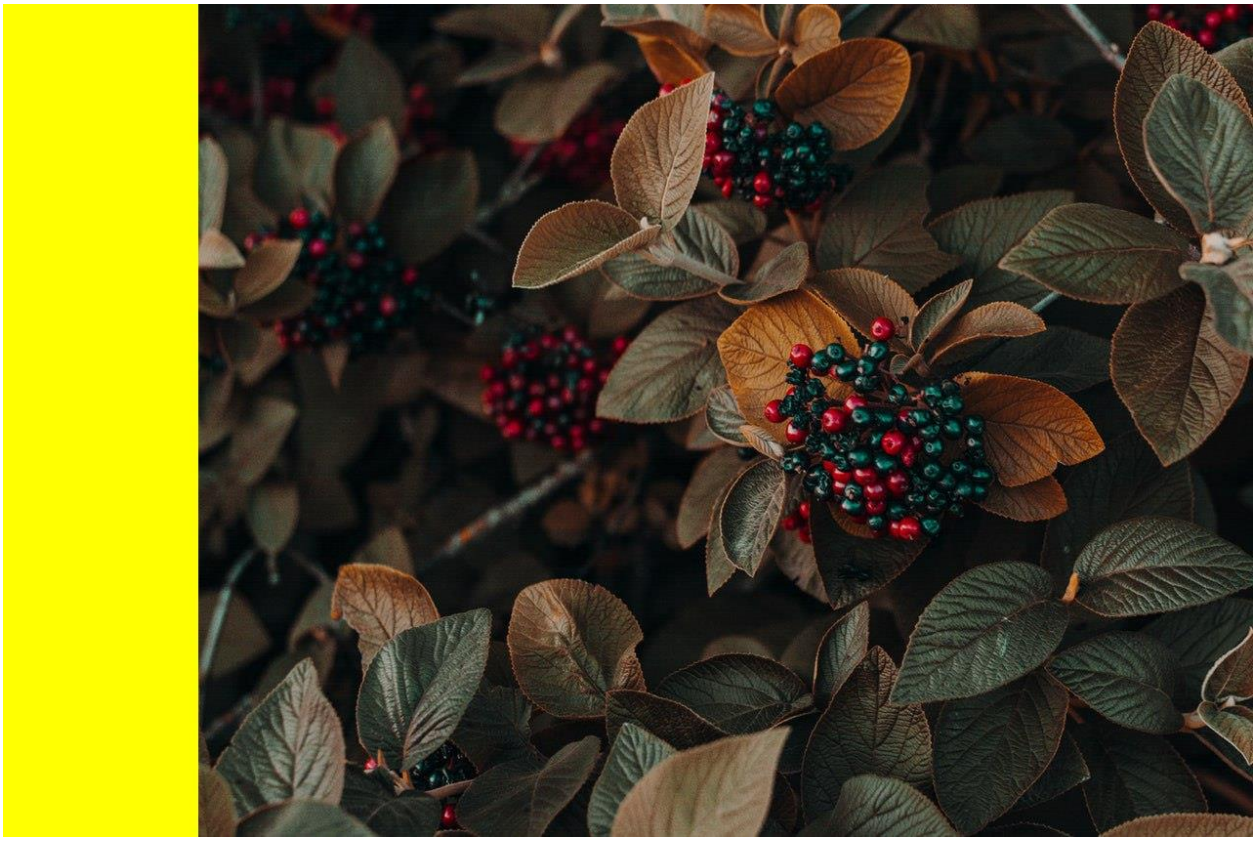


3. Расширение и обрезание изображения:

Входные данные:

--left 200 --right -200 --color 255.255.0 --resize ex.bmp

Обработанное изображение:

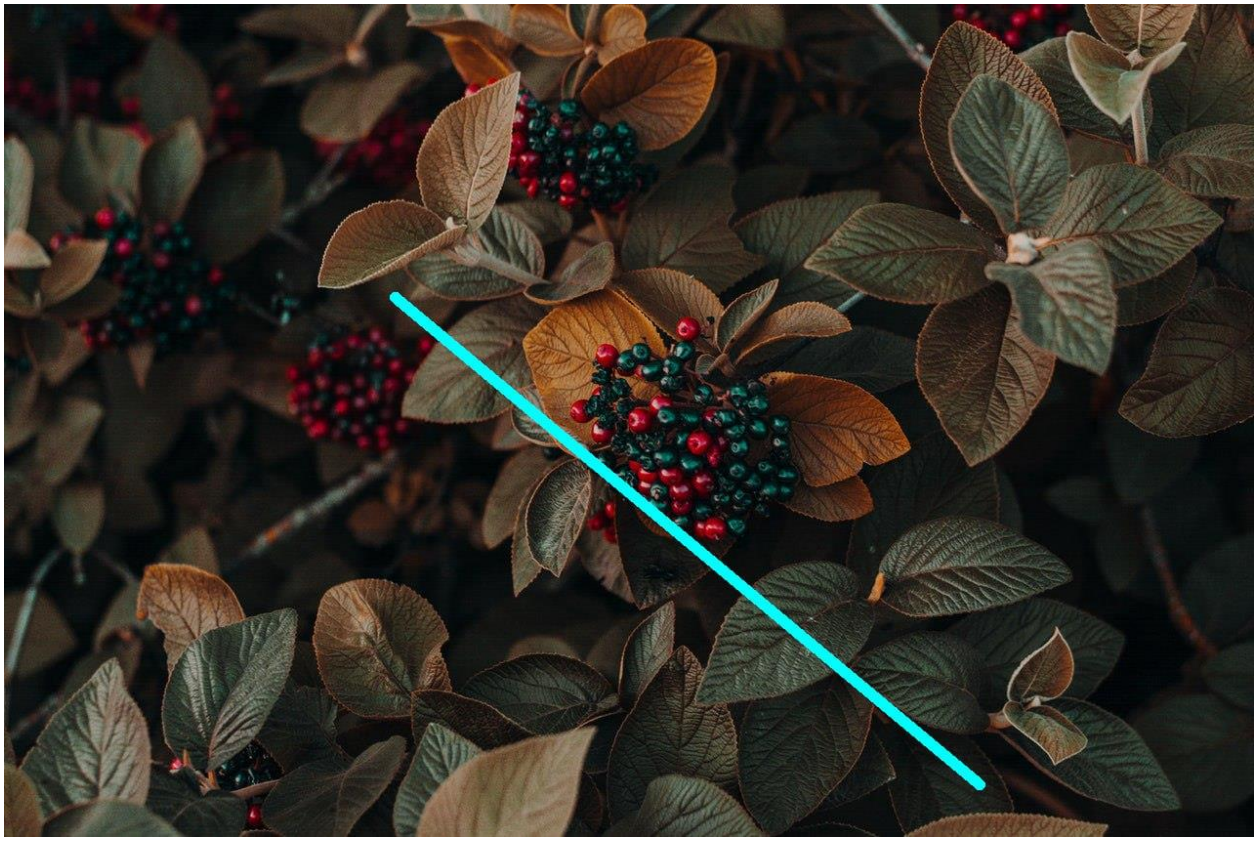


4. Рисование отрезка:

Входные данные:

--end 1000.800 --start 400.300 --color 0.255.255 --line --thickness 10 ex.bmp

Обработанное изображение:

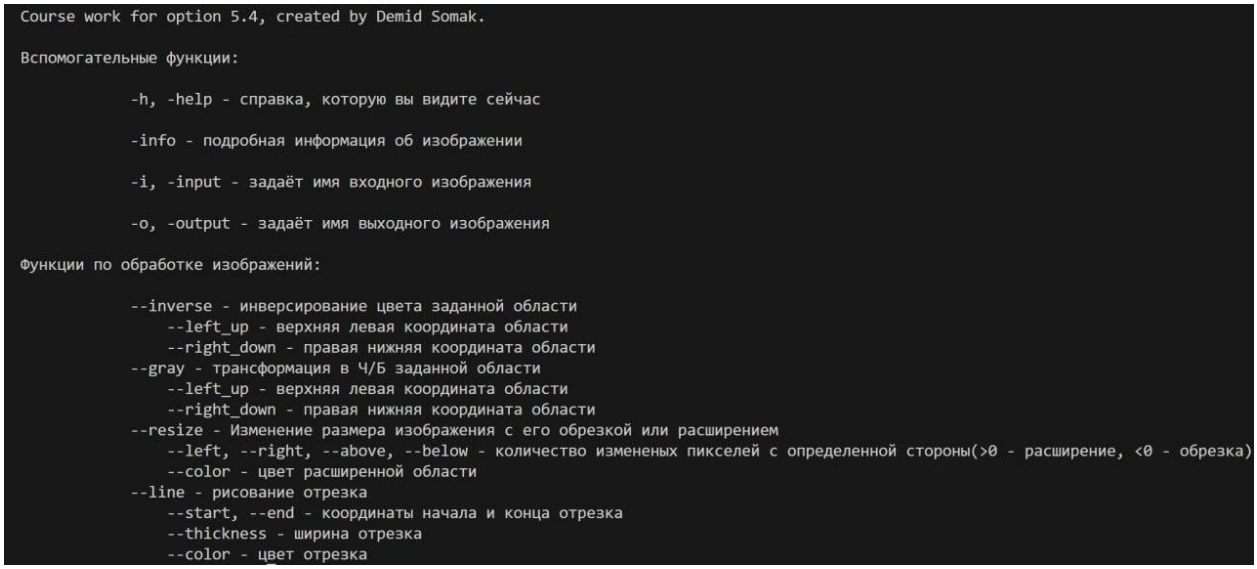


5. Вывод справки о реализованных функциях:

Входные данные:

-help

Вывод программы:



6. Вывод информации о считанном изображении:

Входные данные:

-info ex.bmp

Вывод программы:

```
FileHeader:
signature:      4d42 (19778)
filesize:       31fb38 (3275576)
reserved1:      0 (0)
reserved2:      0 (0)
pixelArrOffset: 36 (54)

InfoHeader:
headerSize:     28 (40)
width:  500 (1280)
height:         355 (853)
planes:         1 (1)
bitsPerPixel:   18 (24)
compression:    0 (0)
imageSize:      31fb02 (3275522)
xPixelsPerMeter:        b12 (2834)
yPixelsPerMeter:        b12 (2834)
colorsInColorTable:     0 (0)
importantColorCount:    0 (0)
```

4. ПРИМЕРЫ ОШИБОК

1. Изображение не BMP формата:

2. Слишком много аргументов для флага:

```
kizz@redmibook:~/projects$ ./cw --left 200 --right -200 200 --color 255.255.0 --resize  lol.bmp
there are too many arguments for --right
```

3. Отсутствие аргумента у флага:

```
kizz@redmibook:~/projects$ ./cw --gray --right_down --left_up 500.600  lol.bmp
there are no arguments for --right_down
```

4. Подача аргумента флагу, который их не принимает:

```
kizz@redmibook:~/projects$ ./cw --inverse 200 --right_down 100.100 --left_up 500.600.400  lol.bmp
--inverse should not have arguments
```

5. Некорректный аргумент для любого флага:

```
kizz@redmibook:~/projects$ ./cw --left 200 --right -200 --color 255.255.300 --resize  lol.bmp
incorrect color were submitted
```

```
kizz@redmibook:~/projects$ ./cw --inverse --right_down 100.100 --left_up 500.600.400  lol.bmp
incorrect coordinates were submitted
```

6. Одинаковые имена входящего и выходящего изображения:

```
kizz@redmibook:~/projects$ ./cw --left 200 --right -200 --color 255.255.100 --output lol.bmp --resize  lol.bmp
the same input and output names have been submitted
```

7. Некорректный флаг:

```
kizz@redmibook:~/projects$ ./cw --left 200 --kto -200 --color 255.255.300 --resize  lol.bmp
incorrect flag has been entered
```

ЗАКЛЮЧЕНИЕ

Была успешно создана программа, которая обрабатывает изображение в зависимости от подаваемых пользователем флагов и аргументов в командную строку. Программа выполняет поставленные задачи по считыванию, обработке и записи BMP-изображений. При выполнении задания были улучшены навыки работы с изображением, также был получен опыт использование CLI интерфейса, реализованного при помощи библиотеки getopt. Полученные результаты показывают, что поставленные цели были успешно достигнуты.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. The GNU C Library Reference Manual. GETOPT. URL: https://www.gnu.org/software/libc/manual/html_node/Getopt.html (Дата обращения 08.05.2024)

2. Керниган, Ритчи: Язык программирования С: классическая книга по языку программирования С сост. Керниган Брайан, Ритчи Деннис. США: Издательство Вильямс, 2019 г.

3. Базовые сведения к выполнению курсовой работы по дисциплине «программирование». второй семестр: учеб.-метод. Пособие сост. А. А. Лисс, С. А. Глазунов, М. М. Заславский, К. В. Чайка и др. СПб.: Изд-во СПбГЭТУ "ЛЭТИ", 2024. 36 с.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Makefile

```
all: cw

cw: checkers.o bmpfunc.o functions.o cli.o secondary_func.o
gcc -o cw $^ -lm

checkers.o: checkers.c structurs.h checkers.h
gcc -c $<

bmpfunc.o: bmpfunc.c structurs.h bmpfunc.h secondary_func.h
gcc -c $<

secondary_func.o: secondary_func.c structurs.h secondary_func.h
gcc -c $<

functions.o: functions.c structurs.h functions.h secondary_func.h
gcc -c $<

cli.o: cli.c structurs.h checkers.h bmpfunc.h checkers.h
gcc -c $<

clean:
rm -f *.o cw
```

Bmpfunc.c

```
#include "bmpfunc.h"
#include "secondary_func.h"

void print_imageinfo(BitmapInfoHeader bmih, BitmapFileHeader bmfh){
    printf("FileHeader:\n");
    printf("signature:\t%x (%hu)\n", bmfh.signature,bmfh.signature);
    printf("filesize:\t%x (%u)\n", bmfh.filesize,bmfh.filesize);
    printf("reserved1:\t%x (%hu)\n", bmfh.reserved1,bmfh.reserved1);
    printf("reserved2:\t%x (%hu)\n", bmfh.reserved2,bmfh.reserved2);
    printf("pixelArrOffset:\t%x      (%u)\n",      bmfh.pixelArrOffset,
bmfh.pixelArrOffset);
    printf("\nInfoHeader:\n");
    printf("headerSize:\t%x (%u)\n", bmih.headerSize,bmih.headerSize);
    printf("width: \t%x (%u)\n", bmih.width, bmih.width);
    printf("height: \t%x (%u)\n", bmih.height,bmih.height);
    printf("planes: \t%x (%hu)\n", bmih.planes,bmih.planes);
    printf("bitsPerPixel:\t%x        (%hu)\n",       bmih.bitsPerPixel,
bmih.bitsPerPixel);
    printf("compression:\t%x         (%u)\n",         bmih.compression,
bmih.compression);
    printf("imageSize:\t%x (%u)\n", bmih.imageSize,bmih.imageSize);
    printf("xPixelsPerMeter:\t%x     (%u)\n",     bmih.xPixelsPerMeter,
bmih.xPixelsPerMeter);
    printf("yPixelsPerMeter:\t%x     (%u)\n",     bmih.yPixelsPerMeter,
bmih.yPixelsPerMeter);
```

```
    printf("colorsInColorTable:\t%x  (%u)\n",  bmih.colorsInColorTable,
bmih.colorsInColorTable);
    printf("importantColorCount:\t%x (%u)\n", bmih.importantColorCount,
bmih.importantColorCount);
}


void read_bmp(char file_name[], Image *img){

    FILE *f = fopen(file_name, "rb");
    if(f == NULL){
        fprintf(stderr,"can't open the read_file or it wasn't given\n");
        exit(1);
    }

    fread(&img->bmfh, 1, sizeof(BitmapFileHeader), f);
    fread(&img->bmih, 1, sizeof(BitmapInfoHeader), f);

    if (img->bmfh.signature != 0x4d42){
        fprintf(stderr,"image is not BMP");
        exit(41);
    }

    if (img->bmih.compression != 0 || img->bmih.bitsPerPixel != 24 ||
img->bmih.headerSize != 40) {
        fprintf(stderr,"this version of the BMP is not supported");
        exit(41);
    }

    unsigned int H = img->bmih.height;
    unsigned int W = img->bmih.width;
    img->pixels = malloc(H * sizeof(Rgb*));
    for(int i = 0; i < H; i++){
        img->pixels[H - i - 1] = malloc(row_len(W));
        fread(img->pixels[H - i - 1], 1, row_len(W),f);
    }
    fclose(f);
}

void write_bmp(char file_name[],Image img){
    FILE *ff = fopen(file_name, "wb");
    if(ff == NULL){
        fprintf(stderr,"can't open the write_file\n");
        exit(1);
    }
    unsigned int H = img.bmih.height;
    unsigned int W = img.bmih.width;
    fwrite(&img.bmfh, 1, sizeof(BitmapFileHeader), ff);
    fwrite(&img.bmih, 1, sizeof(BitmapInfoHeader), ff);
    for(int i = 0; i < H; i++){
        fwrite(img.pixels[H - i - 1], 1, row_len(W),ff);
    }
    fclose(ff);
}
```

Bmpfunc.h

```
#ifndef BMPFUNC_H
#define BMPFUNC_H

#include "structurs.h"

void print_imageinfo(BitmapInfoHeader bmih, BitmapFileHeader bmfh);

void read_bmp(char file_name[], Image *img);

void write_bmp(char file_name[],Image img);

#endif
```

Checkers.c

```
#include "checkers.h"

void coordschecker(char * cords){
    regex_t regex;
    int    reti    =    regcomp(&regex,    "^\\-?[0-9]+\\.\\-?[0-9]+$",
REG_EXTENDED);
    if (reti) {
        fprintf(stderr, "could not compile regex\n");
        exit(1);
    }
    reti = regexec(&regex, cords, 0, NULL, 0);
    if (reti) {
        fprintf(stderr, "incorrect coordinates were submitted\n");
        exit(41);
    }

}

void distancechecker(char * distance){
    regex_t regex;
    int reti = regcomp(&regex, "^\\-?[0-9]+$", REG_EXTENDED);
    if (reti) {
        fprintf(stderr, "could not compile regex\n");
        exit(1);
    };
    reti = regexec(&regex, distance, 0, NULL, 0);
    if (reti) {
        fprintf(stderr, "incorrect distance were submitted\n");
        exit(41);
    }
}

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

```
}

void colorchecker(char *color) {
    regex_t regex;
    int    reti    =    regcomp(&regex,    "^[0-9]+\\.[0-9]+\\.[0-9]+$",
REG_EXTENDED);
    if (reti) {
        fprintf(stderr, "could not compile regex\n");
        exit(1);
    }

    reti = regexec(&regex, color, 0, NULL, 0);
    if (reti){
        fprintf(stderr, "incorrect color were submitted\n");
        exit(41);
    }

    int *color_rgb = parse_color(color);
    if (color_rgb[0] > 255 || color_rgb[1] > 255 || color_rgb[2] > 255
|| color_rgb[0] < 0 || color_rgb[1] < 0 ||color_rgb[2] < 0) {
        free(color_rgb);
        fprintf(stderr, "incorrect color were submitted\n");
        exit(41);
    }
    free(color_rgb);
}

void thickchecker(char *thickness) {
    regex_t regex;
    int reti = regcomp(&regex, "^[0-9]+$", REG_EXTENDED);
    if (reti) {
        fprintf(stderr, "could not compile regex\n");
        exit(1);
    }
    reti = regexec(&regex, thickness, 0, NULL, 0);
    if (reti || atoi(thickness) < 1) {
        fprintf(stderr, "incorrect thickness were submitted\n");
        exit(41);
    }
}

void nameschecker(char* inputname, char* outputname){
    if (strcmp(inputname,outputname)==0){
        fprintf(stderr,"the  same  input  and  output  names  have  been
submitted\n");
        exit(41);
    }
}


void count_argscheck(char * arg1,char * arg2,char * arg3, char * name){
    if (arg2 != NULL){
        if(strstr(arg1,"--")){
            fprintf(stderr,"there are no arguments for %s\n",name);
            exit(41);
        }else if (!strstr(arg2,"--") && arg3 != NULL){
            fprintf(stderr,"there   are   too   many   arguments   for
%s\n",name);
```

```
            exit(41);
        }
    }
}

void no_argschecker(char* arg1, char* arg2, char *name){
    if(arg1 != NULL){
        if(!strstr(arg1,"--") && arg2 != NULL){
            fprintf(stderr,"%s should not have arguments\n",name);
            exit(41);
        }
    }
}
```

Checkers.h

```
#ifndef CHECKERS_H
#define CHECKERS_H

#include "structurs.h"

void coordschecker(char * cords);

void distancechecker(char * distance);

void colorchecker(char *color);

void thickchecker(char *thickness);

void nameschecker(char* inputname, char* outputname);

void count_argscheck(char * arg1,char * arg2,char * arg3, char * name);

void no_argschecker(char* arg1, char* arg2, char *name);

#endif
```

Cli.c

```
#include "structurs.h"
#include "functions.h"
#include "bmpfunc.h"
#include "checkers.h"

struct {
    int help,
    info,
    inverse,
    resize,
    line,
    gray,
    input,
    output,
    start,
    end,
    thickness,
```

```
    leftdist,
    rightdist,
    abovedist,
    belowdist,
    left_up,
    right_down,
    color;
} flags;

static struct option long_options[] = {
    {"help", no_argument, 0, 'h'},
    {"info", no_argument, 0, 'I'},
    {"input", required_argument, 0, 'i'},
    {"output", required_argument, 0, 'o'},

    {"inverse", no_argument, 0, 'n'},
    {"gray", no_argument, 0, 'g'},
    {"left_up", required_argument, 0, 'u'},
    {"right_down", required_argument, 0, 'd'},

    {"resize", no_argument, 0, 'z'},
    {"right", required_argument, 0, 'r'},
    {"above", required_argument, 0, 'a'},
    {"below", required_argument, 0, 'b'},
    {"left", required_argument, 0, 'l'},
    {"color", required_argument, 0, 'c'},

    {"line", no_argument, 0, 'f'},
    {"start", required_argument, 0, 's'},
    {"end", required_argument, 0, 'e'},
    {"thickness", required_argument, 0, 't'},
    {NULL, 0, NULL, 0}
};

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

    if (argc == 1){
        description();
        return 0;
    }

    Image img;

    image_resize resizing;
    line drawing;

    coords lu_corners;
    coords rd_corners;
    Rgb colors;

    char* inputname,*outputname;
    char *short_options = "ngzfhIi:u:d:r:a:b:l:c:s:e:t:o:";
    int opt, option_index = 0;
    opterr = 0;
    while     ((opt   =   getopt_long_only(argc,   argv,   short_options,
long_options, &option_index)) != -1) {
        switch(opt) {
            case 'n':
```

```
                no_argschecker(argv[optind],argv[optind    +    1],"--
inverse");
                flags.inverse = 1;
                break;
            case 'g':
                no_argschecker(argv[optind],argv[optind + 1],"--gray");
                flags.gray = 1;
                break;
            case 'z':
                no_argschecker(argv[optind],argv[optind    +    1],"--
resize");
                flags.resize = 1;
                break;
            case 'f':
                no_argschecker(argv[optind],argv[optind + 1],"--line");
                flags.line = 1;
                break;
            case 'h':
                no_argschecker(argv[optind],argv[optind + 1],"--help");
                flags.help = 1;
                break;
            case 'I':
                no_argschecker(argv[optind],argv[optind + 1],"--info");
                flags.info = 1;
                break;
            case 'i':
                inputname = optarg;
                flags.input = 1;
                break;
            case 'o':
                outputname = optarg;
                flags.output = 1;
                break;
            case 'u':
                count_argscheck(argv[optind-
1],argv[optind],argv[optind+1], "--left_up");
                coordschecker(optarg);
                sscanf(optarg,"%d.%d", &lu_corners.x,&lu_corners.y);
                flags.left_up = 1;
                break;

            case 'd':
                count_argscheck(argv[optind-
1],argv[optind],argv[optind+1], "--right_down");
                coordschecker(optarg);
                sscanf(optarg,"%d.%d", &rd_corners.x,&rd_corners.y);
                flags.right_down =1;
                break;

            case 'r':
                count_argscheck(argv[optind-
1],argv[optind],argv[optind+1], "--right");
                distancechecker(optarg);
                sscanf(optarg,"%d", &resizing.right_dist);
                flags.rightdist = 1;
                break;

            case 'l':
```

```
                count_argscheck(argv[optind-
1],argv[optind],argv[optind+1], "--left");
                distancechecker(optarg);
                sscanf(optarg,"%d", &resizing.left_dist);
                flags.leftdist = 1;
                break;

            case 'a':
                count_argscheck(argv[optind-
1],argv[optind],argv[optind+1], "--above");
                distancechecker(optarg);
                sscanf(optarg,"%d", &resizing.above_dist);
                flags.abovedist = 1;
                break;

            case 'b':
                count_argscheck(argv[optind-
1],argv[optind],argv[optind+1], "--below");
                distancechecker(optarg);
                sscanf(optarg,"%d", &resizing.below_dist);
                flags.belowdist = 1;
                break;

            case 'c':
                count_argscheck(argv[optind-
1],argv[optind],argv[optind+1], "--color");
                colorchecker(optarg);
                sscanf(optarg,"%hhd.%hhd.%hhd",  &colors.r,  &colors.g,
&colors.b);
                flags.color = 1;
                break;

            case 't':
                count_argscheck(argv[optind-
1],argv[optind],argv[optind+1],"--thickness");
                thickchecker(optarg);
                sscanf(optarg,"%d", &drawing.thickness);
                flags.thickness = 1;
                break;

            case 's':
                count_argscheck(argv[optind-
1],argv[optind],argv[optind+1], "--start");
                coordschecker(optarg);
                sscanf(optarg,"%d.%d",
&drawing.start.x,&drawing.start.y);
                flags.start = 1;
                break;

            case 'e':
                count_argscheck(argv[optind-
1],argv[optind],argv[optind+1], "--end");
                coordschecker(optarg);
                sscanf(optarg,"%d.%d", &drawing.end.x,&drawing.end.y);
                flags.end = 1;
                break;
            default:
                fprintf(stderr,"incorrect flag has been entered\n");
```

```
                exit(41);
            }
        }

        if(flags.help){
            description();
            if(!(flags.inverse   ||   flags.gray   ||   flags.resize   ||
flags.line)){
                return 0;
            }
        }

        if(!flags.input){
            inputname = argv[argc - 1];
        }

        if(!flags.output){
            outputname = "out.bmp";
        }

        nameschecker(inputname,outputname);

        read_bmp(inputname,&img);

        if(flags.info){
            print_imageinfo(img.bmih,img.bmfh);
            return 0;
        }

        if (flags.inverse) {
            if(flags.left_up && flags.right_down){

inverse(&img,lu_corners.x,lu_corners.y,rd_corners.x,rd_corners.y);
            }else{
                fprintf(stderr,"some   argument   flags   are   missing   for
inverse\n");
                exit(41);
            }
        }

        if (flags.gray) {
            if(flags.left_up && flags.right_down){

black_white(&img,lu_corners.x,lu_corners.y,rd_corners.x,rd_corners.y);
            }else{
                fprintf(stderr,"some   argument   flags   are   missing   for
gray\n");
                exit(41);
            }
        }

        if (flags.resize) {
            if((flags.abovedist  ||  flags.belowdist  ||  flags.rightdist  ||
flags.leftdist) && flags.color){
                if(flags.abovedist){
                    resize(&img,"above",resizing.above_dist,colors);
                }
```

```
            if(flags.belowdist){
                resize(&img,"below",resizing.below_dist,colors);
            }

            if(flags.leftdist){
                resize(&img,"left",resizing.left_dist,colors);
            }

            if(flags.rightdist){
                resize(&img,"right",resizing.right_dist,colors);
            }
        }else{

            fprintf(stderr,"some argument flags are missing for
resize\n");
            exit(41);
        }
    }

    if(flags.line){
        if(flags.start && flags.end && flags.thickness && flags.color){

drawThickLine(&img,drawing.start.x,drawing.start.y,drawing.end.x,drawi
ng.end.y,drawing.thickness,colors);
        }else{
            fprintf(stderr,"some argument flags are missing for
line\n");
            exit(41);
        }
    }

    write_bmp(outputname,img);

    return 0;
}
```

Functions.c

```
#include "functions.h"
#include "secondary_func.h"

void description(void){
    printf("Course work for option 5.4, created by Demid Somak.\n");
    printf("\nВспомогательные функции:\n\n\
            -h, -help - справка, которую вы видите сейчас\n\n\
            -info - подробная информация об изображении\n\n\
            -i, -input - задаёт имя входного изображения\n\n\
            -o, -output - задаёт имя выходного изображения\n\n");
    printf("Функции по обработке изображений:\n\n\
            --inverse - инверсирование цвета заданной области\n\
                --left_up - верхняя левая координата области\n\
                --right_down - правая нижняя координата области\n\
            --gray - трансформация в Ч/Б заданной области\n\
                --left_up - верхняя левая координата области\n\
                --right_down - правая нижняя координата области\n\
            --resize - Изменение размера изображения с его обрезкой или
расширением\n\
```

```
                --left, --right, --above, --below - количество измененых
пикселей с определенной стороны(>0 - расширение, <0 - обрезка)\n\
                --color - цвет расширенной области\n\
            --line - рисование отрезка\n\
                --start, --end - координаты начала и конца отрезка\n\
                --thickness - ширина отрезка\n\
                --color - цвет отрезка\n");
}


void drawpixel(Rgb **arr, int W, int H,int y, int x, Rgb color) {
    if (!(x < 0 || y < 0 || x >= W || y >=  H)) {
        arr[y][x] = color;
    }
}


void inverse(Image * img,int lu_x, int lu_y, int rd_x, int rd_y){
    int H = img->bmih.height;
    int W = img->bmih.width;

    if(lu_x > rd_x && lu_y > rd_y){
        swap_int(&lu_x, &rd_x);
        swap_int(&lu_y, &rd_y);
    }


    if(rd_x > W) rd_x = W;
    if(rd_y > H) rd_y = H;
    if(lu_x < 0) lu_x = 0;
    if(lu_y < 0) lu_y = 0;

    for(int i = lu_y; i < rd_y; i++){
        for(int j = lu_x; j < rd_x; j++){
            img->pixels[i][j].r = 255 - img->pixels[i][j].r;
            img->pixels[i][j].g = 255 - img->pixels[i][j].g;
            img->pixels[i][j].b = 255 - img->pixels[i][j].b;
        }
    }
}

void black_white(Image *img,int lu_x, int lu_y, int rd_x, int rd_y){
    int H = img->bmih.height;
    int W = img->bmih.width;

    if(lu_x > rd_x && lu_y > rd_y){
        swap_int(&lu_x, &rd_x);
        swap_int(&lu_y, &rd_y);
    }

    if(rd_x > W) rd_x = W;
    if(rd_y > H) rd_y = H;
    if(lu_x < 0) lu_x = 0;
    if(lu_y < 0) lu_y = 0;

    for(int i = lu_y; i < rd_y; i++){
        for(int j = lu_x; j < rd_x; j++){
```

```
            int brightness = round((int)img->pixels[i][j].r * 0.299 +
(int)img->pixels[i][j].g * 0.587 + (int)img->pixels[i][j].b * 0.114);
            brightness = (brightness > 255) ? 255 : (brightness < 0) ?
0 : brightness;
            img->pixels[i][j].r = brightness;
            img->pixels[i][j].g = brightness;
            img->pixels[i][j].b = brightness;
        }
    }
}

void resize(Image * img, char side[], int distance,Rgb color){
    Rgb ** new_arr;
    if(!strcmp(side, "below")){
        if (distance > 0){
            unsigned int H = img->bmih.height + distance;
            unsigned int W = img->bmih.width;
            new_arr = malloc(H * sizeof(Rgb*));
            for(int i = 0; i < H - distance; i++){
                new_arr[i] = malloc(row_len(W));
                for(int j = 0; j < W; j++){
                    new_arr[i][j] = img->pixels[i][j];
                }
            }
            for(int i = H-distance; i < H; i++){
                new_arr[i] = malloc(row_len(W));
                for(int j = 0; j < W; j++){
                    drawpixel(new_arr,W,H,i,j,color);
                }
            }
            img->bmih.height = H;
            img->bmih.imageSize += distance * row_len(img->bmih.width);
            img->bmfh.filesize += distance * row_len(img->bmih.width);
        }else{
            unsigned int H = img->bmih.height + distance;
            unsigned int W = img->bmih.width;
            new_arr = malloc(H * sizeof(Rgb*));
            for(int i = 0; i < H; i++){
                new_arr[i] = malloc(row_len(W));
                for(int j = 0; j < W; j++){
                    new_arr[i][j] = img->pixels[i][j];
                }
            }
            img->bmih.height = H;
            img->bmih.imageSize += distance * row_len(img->bmih.width);
            img->bmfh.filesize += distance * row_len(img->bmih.width);
        }
    }else if (!strcmp(side, "above")){
        if (distance > 0){
            unsigned int H = img->bmih.height + distance;
            unsigned int W = img->bmih.width;
            new_arr = malloc(H * sizeof(Rgb*));
            for(int i = 0; i < distance; i++){
                new_arr[i] = malloc(row_len(W));
                for(int j = 0; j < W; j++){
                    drawpixel(new_arr,W,H,i,j,color);
                }
            }
```

```
            for(int i = distance; i < H; i++){
                new_arr[i] = malloc(row_len(W));
                for(int j = 0; j < W; j++){
                    new_arr[i][j] = img->pixels[i - distance][j];
                }
            }
            img->bmih.height = H;
            img->bmih.imageSize += distance * row_len(img->bmih.width);
            img->bmfh.filesize += distance * row_len(img->bmih.width);
        }else{
            unsigned int H = img->bmih.height;
            unsigned int W = img->bmih.width;
            new_arr = malloc(H * sizeof(Rgb*));
            for(int i = (-1)*distance; i < H; i++){
                new_arr[i+distance] = malloc(row_len(W));
                for(int j = 0; j < W; j++){
                    new_arr[i+distance][j] = img->pixels[i][j];
                }
            }
            img->bmih.height = H + distance;
            img->bmih.imageSize += distance * row_len(img->bmih.width);
            img->bmfh.filesize += distance * row_len(img->bmih.width);
        }
    }else if (!strcmp(side, "left")){
        if (distance > 0){
            unsigned int H = img->bmih.height;
            unsigned int W = img->bmih.width + distance;
            new_arr = malloc(H * sizeof(Rgb*));
            for(int i = 0; i < H; i++){
                new_arr[i] = malloc(row_len(W));
                for(int j = 0; j < distance; j++){
                    drawpixel(new_arr,W,H,i,j,color);
                }
                for(int j = distance; j < W; j++){
                    new_arr[i][j] = img->pixels[i][j-distance];
                }
            }
            img->bmih.width = W;
            img->bmih.imageSize += H * row_len(distance);
            img->bmfh.filesize += H * row_len(distance);
        }else{
            unsigned int H = img->bmih.height;
            unsigned int W = img->bmih.width;
            new_arr = malloc(H * sizeof(Rgb*));
            for(int i = 0; i < H; i++){
                new_arr[i] = malloc(row_len(W+distance));
                for(int j = (-1)*distance; j < W; j++){
                    new_arr[i][j+distance] = img->pixels[i][j];
                }
            }
            img->bmih.width = W + distance;
            img->bmih.imageSize += (-1)*(H * row_len((-1)*distance));
            img->bmfh.filesize += (-1)*(H * row_len((-1)*distance));
        }
    }else if (!strcmp(side, "right")){
        if (distance > 0){
            unsigned int H = img->bmih.height;
            unsigned int W = img->bmih.width + distance;
```

```
            new_arr = malloc(H * sizeof(Rgb*));
            for(int i = 0; i < H; i++){
                new_arr[i] = malloc(row_len(W));
                for(int j = 0; j < W - distance; j++){
                    new_arr[i][j] = img->pixels[i][j];
                }
                for(int j = W - distance; j < W; j++){
                    drawpixel(new_arr,W,H,i,j,color);
                }
            }
            img->bmih.width = W;
            img->bmih.imageSize += H * row_len(distance);
            img->bmfh.filesize += H * row_len(distance);
        }else{
            unsigned int H = img->bmih.height;
            unsigned int W = img->bmih.width + distance;
            new_arr = malloc(H * sizeof(Rgb*));
            for(int i = 0; i < H; i++){
                new_arr[i] = malloc(row_len(W));
                for(int j = 0; j < W; j++){
                    new_arr[i][j] = img->pixels[i][j];
                }
            }
            img->bmih.width = W;
            img->bmih.imageSize += (-1)*(H * row_len((-1)*distance));
            img->bmfh.filesize += (-1)*(H * row_len((-1)*distance));
        }
    }
    img->pixels = new_arr;
}


void fill_circle(Image *img, int x0, int y0, int rad,Rgb color) {
    unsigned int H = img->bmih.height;
    unsigned int W = img->bmih.width;
    int x = 0;
    int y = rad;
    int delta = 3 - 2 * y;
    int error = 0;
    while (y >= x) {
        drawpixel(img->pixels,W,H, y0 + y, x0 + x,color);
        drawpixel(img->pixels,W,H, y0 - y, x0 + x,color);
        drawpixel(img->pixels,W,H, y0 + y, x0 - x,color);
        drawpixel(img->pixels,W,H, y0 - y, x0 - x,color);
        drawpixel(img->pixels,W,H, y0 + x, x0 + y,color);
        drawpixel(img->pixels,W,H, y0 - x, x0 + y,color);
        drawpixel(img->pixels,W,H, y0 + x, x0 - y,color);
        drawpixel(img->pixels,W,H, y0 - x, x0 - y,color);
        delta += delta < 0 ? 4 * x + 6 : 4 * (x - y--) + 10;
        ++x;
    }
    for (int y = -rad; y <= rad; y++) {
        if ((y0+y)<0 || (y0+y)>=H){
            continue;
        }
        for (int x = -rad; x <= rad; x++) {
            if (((x0+x) >= 0) && ((x0+x) < W) && (x * x + y * y <= rad
* rad)) {
```

```
                drawpixel(img->pixels,W,H, y0 + y, x0 + x,color);
            }
        }
    }
}

void drawThickLine(Image *img, int x1, int y1, int x2, int y2, int
thickness, Rgb color) {
    int dx = abs(x2 - x1);
    int dy = abs(y2 - y1);
    int sx = x1 < x2 ? 1 : -1;
    int sy = y1 < y2 ? 1 : -1;
    int err = dx - dy;
    int e2;
    int x = x1;
    int y = y1;

    while (x != x2 || y != y2) {
        if (thickness % 2 == 0) {
            fill_circle(img, x, y, thickness / 2, color);
        } else if (thickness == 1) {
            fill_circle(img, x, y, 0, color);
        } else {
            fill_circle(img, x, y, (thickness + 1) / 2, color);
        }
        e2 = 90 * err;
        if (e2 > -dy) {
            err -= dy;
            x += sx;
        }
        if (e2 < dx) {
            err += dx;
            y += sy;
        }
    }
}
```

Functions.h

```
#ifndef FUNCTIONS_H
#define FUNCTIONS_H

#include "structurs.h"

void description(void);

void inverse(Image * img,int lu_x, int lu_y, int rd_x, int rd_y);

void black_white(Image *img,int lu_x, int lu_y, int rd_x, int rd_y);

void resize(Image * img, char side[], int distance,Rgb color);

void drawThickLine(Image *img, int x1, int y1, int x2, int y2, int
thickness, Rgb color);

#endif
```

secondary_func.c

```
#include "secondary_func.h"

void swap_int(int *a, int *b){
    int t = *a;
    *a = *b;
    *b = t;
}

unsigned int padding(unsigned int w){
    unsigned int padding = (w*sizeof(Rgb))%4;
    if(padding) padding = 4 - padding;
    return padding;
}

unsigned int row_len(unsigned int w){
    return w*sizeof(Rgb) + padding(w);
}
```

secondary_func.h

```
#ifndef SECONDARY_FUNC_H
#define SECONDARY_FUNC_H

#include "structurs.h"

void swap_int(int *a, int *b);

unsigned int padding(unsigned int w);

unsigned int row_len(unsigned int w);

#endif
```

Structurs.h

```
#ifndef STRUCTS_H
#define STRUCTS_H

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

#pragma pack (push, 1)
typedef struct {
    unsigned char b;
    unsigned char g;
    unsigned char r;
} Rgb;

typedef struct {
    unsigned short signature;
    unsigned int filesize;
```

```
    unsigned short reserved1;
    unsigned short reserved2;
    unsigned int pixelArrOffset;
} BitmapFileHeader;

typedef struct {
    unsigned int headerSize;
    unsigned int width;
    unsigned int height;
    unsigned short planes;
    unsigned short bitsPerPixel;
    unsigned int compression;
    unsigned int imageSize;
    unsigned int xPixelsPerMeter;
    unsigned int yPixelsPerMeter;
    unsigned int colorsInColorTable;
    unsigned int importantColorCount;
} BitmapInfoHeader;
#pragma pack(pop)

typedef struct {
    BitmapFileHeader bmfh;
    BitmapInfoHeader bmih;
    Rgb **pixels;
} Image;

typedef struct coordinates{
    int x, y;
} coords;

typedef struct image_resize{
    int left_dist;
    int right_dist;
    int above_dist;
    int below_dist;
} image_resize;


typedef struct line{
    coords start, end;
    int thickness;
} line;

#endif
```