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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка BMP файла

Студент гр. 3341 _____ Чинаева М.Р.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2024

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Чинаева М. Р.

Группа 3341

Вариант 4.

Тема работы: Обработка 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. 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

2. 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

3. 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)

4. 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
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
Все подзадачи, ввод/вывод должны быть реализованы в виде отдельной функции.

Содержание пояснительной записки:
разделы «Аннотация», «Содержание», «Введение», «Ход работы», «Пример работы программы», «Заключение», «Список использованных источников»

Предполагаемый объем пояснительной записки:
Не менее 15 страниц.

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 21.05.2024

Дата защиты реферата: 23.05.2024

Студент		Чинаева М.Р.
Преподаватель		Глазунов С.А.

АННОТАЦИЯ

В данной курсовой работе была реализована программа, обрабатывающая BMP изображения, не имеющие сжатия, с глубиной 24 бита. Программа проверяет тип изображения, его версию, при соответствии требованиям в дальнейшем обрабатывает его и подаёт на выход изменённую копию изображения. Взаимодействие с программой осуществляется с помощью CLI (интерфейс командной строки).

SUMMARY

In this course has been created a program that processes uncompressed BMP images with a depth of 24 bits. The program checks the type of image, its version, if it meets the requirements, it further processes it and outputs a modified copy of the image. Interaction with the program is performed using CLI (command line interface).

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной работы является создание программы на языке Си для обработки BMP изображений.

Для достижения поставленной цели потребовалось решить ряд задач:

- изучить, как устроены BMP файлы, что они в себе содержат;

- научиться считывать и записывать BMP изображения;

- разработать функцию инвертирования заданной области;

- разработать функцию преобразования в ч/б формат заданной области;

- разработать функцию изменения размера изображения с заданными параметрами;

- разработать функцию рисования отрезка по заданным координатам с заданной толщиной;

- изучить библиотеку *getopt.h*;

- научиться работать с аргументами командной строки, длинными и короткими флагами;

- создать *Makefile* для сборки программы;

- протестировать разработанную программу.

1. РАБОТА С ФАЙЛАМИ

1.1. Проверка файла

void check_bmp(const char file_name[], BitmapFileHeader *bmfh, BitmapInfoHeader *bmif)

Перед получением информации из файла производится проверка на соответствие формату BMP и прочим требуемым параметрам. Открывается файл, сохраняется указатель на файл, затем с помощью fread структуры заполняются информацией из файла.

Сообщение об ошибки выводится в случаях:

1. файл не открывается
2. файл не соответствует формату bmp
3. у файла есть сжатие
4. у файла не 24 бита на пиксель

В ином случае происходит закрытие файла.

1.2. Чтение файла

Rgb** read_bmp(const char file_name[], BitmapFileHeader* bmfh, BitmapInfoHeader* bmif)

Функция открывает файл для чтения в бинарном режиме, считывает информацию об изображении в предоставленные структуры. Вычисляется размер выравнивания. После этого функция выделяет память для двумерного массива пикселей изображения. Функция считывает каждую строку пикселей изображения с последней строки, так как строки хранятся снизу-вверх. Файл закрывается.

Функция возвращает указатель на считанный массив.

1.3 Запись информации в файл

void write_bmp(const char new_file[], BitmapFileHeader *bmfh, BitmapInfoHeader *bmif, Rgb **arr)

Функция открывает файл для записи в бинарном режиме и записывает туда информацию об изображении. По аналогии с функцией чтения, вычисляется

количество байтов заполнения для выравнивания строк. Функция проходит через все строки изображения (так же начиная с последней) и записывает данные в файл. Файл закрывается.

1.4 Консольный интерфейс

void get_options(int argc, char* argv[])

Консольный интерфейс реализован при помощи библиотеки getopt.

Если аргументы не были введены, выводится сообщение об ошибке. Создается массив структур option, в котором перечислены длинные флаги, соответствующие им короткие флаги. Далее инициализируются все необходимые переменные. Далее с помощью цикла while ((option = getopt_long(argc, argv, "ngrlhIi:o:u:d:L:R:A:B:C:s:e:t:", long_options, NULL)) != -1) поочередно считываются флаги. Если после флага следуют аргументы, то они проверяются на соответствие необходимому формату. В случае несовпадении программа завершает работу с ошибкой. Если флаг неизвестен, то программа также завершает работу с ошибкой.

Далее обрабатываются и проверяются введенные флаги. Если аргументов не хватает, флаг был введен более одного раза, не был введен ни один флаг, или вызвано сразу несколько функций, то программа завершает работу с ошибкой.

Считывается и проверяется bmp файл. Далее вызывается соответствующая функция. После освобождается память из-под выделенных структур и двумерного массива пикселей.

2. ВСПОМОГАТЕЛЬНЫЕ ФУНКЦИИ

2.1 Проверка цветовых компонентов

Rgb check_color(int r, int g, int b)

Если компонент цвета лежат не в диапазоне 0…255, то функция возвращает ошибку. Если все хорошо, то создаётся экземпляр структуры Rgb.

Функция возвращает созданный цвет.

2.2. Проверка координат области

void check_coord(BitmapInfoHeader *bmif, int *left_x, int *up_y, int *right_x, int *down_y)

Если левая координата x больше правой координаты, то их численные значения меняются. Если верхняя координата y больше нижней координаты, то их численные значения меняются.

Функция ничего не возвращает.

2.3. Инвертирование цвета

void inverse_color(Rgb *color)

Каждая компонента меняется на инвертированную. Функция ничего не возвращает.

2.4. Перевод цвета в ч/б вариант

void gray_color(Rgb *color)

По формуле вычисляется значение серого, зачем каждой компоненте присваивается это значение. Функция ничего не возвращает.

2.5. Закрашивание пикселя

void color_pixel(Rgb *pixel, Rgb color)

У переданного пикселя меняются компоненты на соответствующие введенному цвету. Функция ничего не возвращает.

2.7. Закрашивание bmp

void color_bmp (Rgb **arr, BitmapInfoHeader* bmif, Rgb new_color)

С помощью функции color_pixel(&arr[y][x], new_color) закрашивается каждый пиксель изображения. Функция ничего не возвращает.

2.8. Проверка пикселя

void color_pixel_check(Rgb** arr, BitmapInfoHeader* bmif, int x, int y, Rgb color)

Если координаты пикселя находятся внутри изображения, то пиксель закрашивается с помощью функции color_pixel(&arr[y][x], color). Функция ничего не возвращает.

2.9. Рисование круга с толщиной 2

void fill_wide_circle(Rgb** arr, BitmapInfoHeader* bmif, int x_center, int y_center, int radius, Rgb color)

С помощью цикла while (y >= x) и алгоритма Брезенхема рисуется окружность по координатам. Пиксели раскрашиваются с помощью color_pixel_check.

2.10. Вывод информации о файле

void print_file_header(BitmapFileHeader header)

void print_info_header(BitmapInfoHeader header)

Функции выводят информацию из структур BitmapFileHeader и BitmapInfoHeader на экран. Функции ничего не возвращает.

2.11. Печать справки

void print_help()

Функция последовательно вводит информацию о флагах и функциях. Функция ничего не возвращает

3. ОСНОВНЫЕ ФУНКЦИИ

3.1. Инвертирование цвета в заданной области

void inverse(Rgb **arr, BitmapInfoHeader bmif, int left_x, int up_y, int right_x, int down_y)

С помощью вложенных циклов for (int y = up_y; y < down_y; y++) и for (int x = left_x; x< right_x; x++) функция проверяет каждый пиксель области и, если он лежит внутри изображения, инвертирует пиксель с помощью функции inverse_color(&arr[y][x]). Функция ничего не возвращает.

3.2. Преобразование в ч/б вид заданную область

void gray(Rgb **arr, BitmapInfoHeader bmif, int left_x, int up_y, int right_x, int down_y)

С помощью вложенных циклов for (int y = up_y; y < down_y; y++) и for (int x = left_x; x< right_x; x++) функция проверяет каждый пиксель области и, если он лежит внутри изображения, переводит его цвет в ч/б с помощью функции gray_color(&arr[y][x]). Функция ничего не возвращает.

3.3. Изменение размера изображения

Rgb** resize(Rgb **arr, BitmapInfoHeader* bmif, int left, int right, int below, int above, Rgb color)

Сначала вычисляются новые значения высоты и ширины. Они сохранются в структуру BitmapInfoHeader для дальнейшей работы. Далее создается новый двумерный массив пикселов (способ аналогичен функции read_bmp). Полученный массив закрашивается в цвет фона с помощью функции color_bmp. Далее с помощью вложенных циклов по осям y и x в соответствие со смещением слева и снизу переносятся пиксели исходного изображения.

Функция возвращает указатель на новый массив пикселей.

3.4. Рисование линнии

void draw_line(Rgb **arr, BitmapInfoHeader* bmif, int start_x, int start_y, int end_x, int end_y, int thickness, Rgb color)

Для вычисления координат линии используется алгоритм Брезенхема. Он реализуется с помощью цикла while (1). В каждой точке, соответствующей линии рисуется окружность с шириной два и диаметром, равным толщине заданной линии, это нужно для того чтобы внутри линии не оставалось не закрашенных пикселей. Выход из цикла происходит, когда координаты начала, изменяющиеся на каждой итерации цикла, становятся равным координатам конца. Функция ничего не возвращает.

ЗАКЛЮЧЕНИЕ

Разработана программа на языке программирования Си, обрабатывающая BMP изображения и имеющая CLI. В ходе выполнения работы было изучено устройство BMP файлов; изучены методы считывание и записи файлов; получены навыки обработки изображений; разработаны функции для инвертирования заданной области; преобразования в ч/б формат заданной области; изменения размера изображения с заданными параметрами; рисования отрезка по заданным координатам с заданной толщиной; изучена библиотека *getopt.h;* изучена работа с аргументами командной строки.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1.https://ru.wikipedia.org/wiki/BMP - структура файла BMP;

2.https://se.moevm.info/lib/exe/fetch.php/courses:programming:programming_cw_metoda_2nd_course_last_ver.pdf.pdf - методические материалы для написания курсовой работы

3.https://www.r5.org/files/books/computers/languages/c/kr/Brian_Kernighan_Dennis_Ritchie-The_C_Programming_Language-RU.pdf – язык программирования Си

4. https://habr.com/ru/articles/55665/ - принцип работы getopt_long

5. https://ru.wikipedia.org/wiki/Алгоритм_Брезенхэма - алгоритм Брезенхэма

6. https://www.baeldung.com/cs/bresenhams-line-algorithm - оптимизированный код для алгоритма Брезенхэма

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

```
#include "option.h"

int main(int argc, char* argv[]) {
    printf("Course work for option 5.4, created by Margarita Chinaeva\n");
    get_options(argc, argv);
    return 0;
}
```

Название файла: bmp.c

```
#include "bmp.h"

void check_bmp(const char file_name[], BitmapFileHeader *bmfh, BitmapInfoHeader
*bmif)
{
    FILE *file = fopen(file_name, "rb");
    fread(bmfh, 1, sizeof(BitmapFileHeader), file);
    fread(bmif, 1, sizeof(BitmapInfoHeader), file);
    if (!file){
        fprintf(stderr, "File Error: file could not be opened\n");
        fclose(file);
        exit(43);
    }
    if (bmfh->signature != 0x4d42){
        fprintf(stderr, "File Error: file must be bmp\n");
        fclose(file);
        exit(43);
    }
    if (bmif->compression != 0){
        fprintf(stderr, "File Error: file must be uncompressed\n");
        fclose(file);
        exit(43);
    }
    if (bmif->bitsPerPixel != 24){
        fprintf(stderr, "File Error: file must have 24 bits per pixel\n");
        fclose(file);
        exit(43);
    }
    fclose(file);
}

Rgb** read_bmp(const char file_name[], BitmapFileHeader* bmfh, BitmapInfoHeader*
bmif) {
    FILE* file = fopen(file_name, "rb");
    fread(bmfh, 1, sizeof(BitmapFileHeader), file);
    fread(bmif, 1, sizeof(BitmapInfoHeader), file);
    unsigned int Height = bmif->height;
    unsigned int Width = bmif->width;
    unsigned int padding = (Width*sizeof(Rgb)) % 4;
    if (padding) padding = 4 - padding;
    Rgb** arr = (Rgb**)calloc(Height, sizeof(Rgb*));
    for (int i = Height - 1; i>=0 ; i--) {
        arr[i] = (Rgb *) calloc(Width*sizeof(Rgb)+padding, sizeof(Rgb));
        fread(arr[i], 1, Width*sizeof(Rgb) + padding, file);
    }
    fclose(file);
```

```
    return arr;
}

void write_bmp(const char new_file[], BitmapFileHeader *bmfh, BitmapInfoHeader
*bmif, Rgb **arr) {
    FILE *file = fopen(new_file, "wb");
    fwrite(bmfh, 1, sizeof(BitmapFileHeader), file);
    fwrite(bmif, 1, sizeof(BitmapInfoHeader), file);
    unsigned int Height = bmif->height;
    unsigned int Width = bmif->width;
    size_t padding = (4 - (Width * sizeof(Rgb)) % 4) % 4;
    for (int i = Height - 1; i>=0 ; i--) {
        fwrite(arr[i], 1, Width * sizeof(Rgb) + padding, file);
    }
    fclose(file);
}
```

Название файла: bmp.h

```
#pragma once

#include "struct.h"
#include <stdlib.h>
#include <stdio.h>

void check_bmp(const char file_name[], BitmapFileHeader *bmfh, BitmapInfoHeader
*bmif);
Rgb** read_bmp(const char file_name[], BitmapFileHeader* bmfh, BitmapInfoHeader*
bmif);
void write_bmp(const char new_file[], BitmapFileHeader *bmfh, BitmapInfoHeader
*bmif, Rgb **arr);
```

Название файла: drawing.c

```
#include "drawing.h"

Rgb check_color(int r, int g, int b){
    if(r>255 || g>255 || b>255 || r<0 || g<0 || b<0){
        fprintf(stderr, "Value Error: component values must be in the range 0 to
255,\n");
        exit(42);
    }
    Rgb color;
    color.r = r;
    color.g = g;
    color.b = b;
    return color;
}

void check_coord(BitmapInfoHeader *bmif, int *left_x, int *up_y, int *right_x,
int *down_y) {
    int Height = bmif->height;
    int Width = bmif->width;
    if (*left_x > *right_x) {
        int tmp = *left_x;
      *left_x = *right_x;
      *right_x = tmp;
    }
    if (up_y > down_y) {
      int tmp = *up_y;
      *up_y = *down_y;
      *down_y = tmp;
    }
```

```
}

void inverse_color(Rgb *color){
    color->r = 255 - color->r;
    color->g = 255 - color->g;
    color->b = 255 - color->b;
}

void inverse(Rgb **arr, BitmapInfoHeader bmif, int left_x, int up_y, int
right_x, int down_y){
    for (int y = up_y; y < down_y; y++){
        for (int x = left_x; x< right_x; x++){
            if ( 0<= y && y < bmif.height && 0<=x && x< bmif.width) {
                inverse_color(&arr[y][x]);
            }
        }
    }
}

void gray_color(Rgb *color){
    int grey = round(0.299*color->r + 0.587*color->g + 0.114 *color->b);
    color->r = grey;
    color->g = grey;
    color->b = grey;
}

void gray(Rgb **arr, BitmapInfoHeader bmif, int left_x, int up_y, int right_x,
int down_y){
    for (int y = up_y; y < down_y; y++){
        for (int x = left_x; x< right_x; x++){
            if ( 0<= y && y < bmif.height && 0<=x && x< bmif.width) {
                gray_color(&arr[y][x]);
            }
        }
    }
}

void color_pixel(Rgb *pixel, Rgb color){
    pixel->b = color.b;
    pixel->g = color.g;
    pixel->r = color.r;
}

void color_bmp (Rgb **arr, BitmapInfoHeader* bmif, Rgb new_color){
    for (int y = 0; y < bmif->height; y++){
        for (int x = 0; x < bmif->width; x++){
            color_pixel(&arr[y][x], new_color);
        }
    }
}

Rgb** resize(Rgb **arr, BitmapInfoHeader* bmif, int left, int right, int below,
int above, Rgb color){
    int old_H = bmif->height;
    int old_W = bmif->width;
    int new_H = old_H + above + below;
    int new_W = old_W + left + right;
    bmif->height = new_H;
    bmif->width = new_W;
    unsigned int padding = (new_H*sizeof(Rgb)) % 4;
    if (padding) padding = 4 - padding;
    Rgb** new_arr = (Rgb**)calloc(new_H, sizeof(Rgb*));
```

```
    for (int y = new_H - 1; y>=0 ; y--) {
        new_arr[y] = (Rgb *) calloc(new_W*sizeof(Rgb)+padding, sizeof(Rgb));
    }
    color_bmp(new_arr, bmif, color);
    for (int y = 0; y < old_H; y++){
        for (int x = 0; x < old_W; x++){
            if ( y+above >= 0 && y+above < new_H && x+left >= 0 && x+left <
new_W){
                new_arr[y+above][x+left] = arr[y][x];
            }
        }
    }
    return new_arr;
}

void color_pixel_check(Rgb** arr, BitmapInfoHeader* bmif, int x, int y, Rgb
color){
    if (x >= 0 && x<bmif->width && y >= 0 && y<bmif->height){
        color_pixel(&arr[y][x], color);
    }
}

void fill_wide_circle(Rgb** arr, BitmapInfoHeader* bmif, int x_center, int
y_center, int radius, Rgb color){
    int x = 0;
    int y = radius;
    int delta = 1 - 2*radius;
    int error = 0;
    while (y >= x){
        color_pixel_check(arr, bmif, x_center + x, y_center + y, color);
        color_pixel_check(arr, bmif, x_center - x, y_center + y, color);
        color_pixel_check(arr, bmif, x_center + x, y_center - y, color);
        color_pixel_check(arr, bmif, x_center - x, y_center - y, color);

        color_pixel_check(arr, bmif, x_center + y, y_center + x, color);
        color_pixel_check(arr, bmif, x_center - y, y_center + x, color);
        color_pixel_check(arr, bmif, x_center + y, y_center - x, color);
        color_pixel_check(arr, bmif, x_center - y, y_center - x, color);

        color_pixel_check(arr, bmif, x_center + x, y_center + y - 1, color);
        color_pixel_check(arr, bmif, x_center - x, y_center + y - 1, color);
        color_pixel_check(arr, bmif, x_center + x, y_center - y + 1, color);
        color_pixel_check(arr, bmif, x_center - x, y_center - y + 1, color);

        color_pixel_check(arr, bmif, x_center + y - 1, y_center + x, color);
        color_pixel_check(arr, bmif, x_center - y + 1, y_center + x, color);
        color_pixel_check(arr, bmif, x_center + y - 1, y_center - x, color);
        color_pixel_check(arr, bmif, x_center - y + 1, y_center - x, color);

        error = 2 * (delta + y) - 1;
        if ((delta < 0) && (error <= 0)){
            delta += 2 * ++x + 1;
            continue;
        }
        if ((delta > 0) && (error > 0)){
            delta -= 2 * --y + 1;
            continue;
        }
        delta += 2 * (++x - --y);
    }
}
```

```
void draw_line(Rgb **arr, BitmapInfoHeader* bmif, int start_x, int start_y, int
end_x, int end_y, int thickness, Rgb color) {
    int delta_x = abs(end_x - start_x);
    int delta_y = abs(end_y - start_y);
    int step_x;
    if (start_x < end_x){
        step_x = 1;
    } else {
        step_x = -1;
    }
    int step_y;
    if (start_y < end_y) {
        step_y = 1;
    } else {
        step_y = -1;
    }
    int val = delta_x-delta_y;

    while (1) {
        int coord_x = start_x - thickness / 2;
        int coord_y = start_y - thickness / 2;
        if (coord_x+thickness>=0 || coord_y+thickness>=0 || coord_x < bmif-
>width || coord_y < bmif->height){
            fill_wide_circle (arr, bmif, start_x, start_y, thickness/2, color);
        }
        if (start_x == end_x && start_y == end_y) break;
        int double_val = 2 * val;
        if (double_val > -delta_y) {
            val -= delta_y;
            start_x += step_x;
        }
        if (double_val < delta_x) {
            val += delta_x;
            start_y += step_y;
        }
    }

}
```

Название файла: drawing.h

```
#pragma once

#include "struct.h"
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

Rgb check_color(int r, int g, int b);
void check_coord(BitmapInfoHeader *bmif, int *left_x, int *up_y, int *right_x,
int *down_y);
void inverse_color(Rgb *color);
void inverse(Rgb **arr, BitmapInfoHeader bmif, int left_x, int up_y, int
right_x, int down_y);
void gray_color(Rgb *color);
void gray(Rgb **arr, BitmapInfoHeader bmif, int left_x, int up_y, int right_x,
int down_y);
void color_pixel(Rgb *pixel, Rgb color);
void color_bmp (Rgb **arr, BitmapInfoHeader* bmif, Rgb new_color);
Rgb** resize(Rgb **arr, BitmapInfoHeader* bmif, int left, int right, int below,
int above, Rgb color);
```

```
void color_pixel_check(Rgb** arr, BitmapInfoHeader* bmif, int y, int x, Rgb
color);
void fill_wide_circle(Rgb** arr, BitmapInfoHeader* bmif, int x_center, int
y_center, int radius, Rgb color);
void draw_line(Rgb **arr, BitmapInfoHeader* bmif, int start_x, int start_y, int
end_x, int end_y, int thickness, Rgb color);
```

Название файла: help.c

```
#include "help.h"

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

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

void print_help(){
      printf("Flags -o, --output are used to set name of output file (out.bmp by
default).\n");
      printf("Flags -i, --input are used to set name of input file (last
argument by default).\n");
      printf("Flag --info is used to get the info about bmp image.\n");
      printf("Functions to edit the image:\n");

      printf("\nFlag --inverse is used to invert a given area. For initializing
it you need to set some values:\n");
      printf("\t--left_up <left.up>: \n\t left is the x coordinate, up is the y
coordinate\n");
      printf("\t--right_down <right.down>: \n\t right is the x coordinate, down
is the y coordinate\n");

      printf("\nFlag --gray is used to gray a given area. For initializing it
you need to set some values:\n");
      printf("\t--left_up <left.up>: \n\t left is the x coordinate, up is the y
coordinate\n");
      printf("\t--right_down <right.down>: \n\t right is the x coordinate, down
is the y coordinate\n");
```

```
      printf("\nFlag --resize is used to resize the image. For initializing it
you need to set some values:\n");
      printf("\t--left <left>\n");
      printf("\t--right <right>\n");
      printf("\t--above <above>\n ");
      printf("\t--below <below>\n ");
      printf("\t You must enter at least one side value. A positive number means
expansion, a negative number means pruning\n");
      printf("\t--color <rrr.ggg.bbb>: \n\t Background color\n");

      printf("\n--line is used to draw line. For initializing it you need to set
some values:\n");
      printf("\t--start <x.y>\n ");
      printf("\t--end <x.y>\n ");
      printf("\t--color <rrr.ggg.bbb>: \n\t Background color\n");
      printf("\t-- thickness <value>: \n\t Value must be positive\n");
}
```

Название файла: help.h

```
#pragma once

#include <stdio.h>
#include "struct.h"

void print_file_header(BitmapFileHeader header);
void print_info_header(BitmapInfoHeader header);
void print_help();
```

Название файла: option.c

```
#include "option.h"

check_option do_check(){
    check_option check;
    check.above = 0;
    check.below = 0;
    check.color = 0;
    check.end = 0;
    check.gray = 0;
    check.input = 0;
    check.inverse = 0;
    check.left = 0;
    check.left_up = 0;
    check.line = 0;
    check.output = 0;
    check.resize = 0;
    check.right = 0;
    check.right_down = 0;
    check.start = 0;
    check.thickness = 0;
    check.info = 0;
    return check;
}

void check_flags(check_option check){
    int sum_check = check.inverse + check.gray + check.resize + check.line +
check.info;
    if (sum_check > 1){
        fprintf(stderr, "Input Error: impossible use several functions at
once\n");
        exit(41);
    }
```

```
    if (sum_check == 0){
        fprintf(stderr, "Input Error: none of the action functions were
called\n");
        exit(41);
    }
    if (check.above>1 || check.below>1 || check.color>1 || check.end>1 ||
check.gray>1 || check.input>1 || check.inverse>1 || check.left>1 ||
check.left_up>1 || check.line>1 || check.output>1 || check.resize>1 ||
check.right>1 || check.right_down>1 || check.start>1 || check.thickness>1){
        fprintf(stderr, "Input Error: forbidden use the same flag multiple
times\n");
        exit(41);
    }
    if (check.info==0 && (check.left_up + check.right_down)!=2 &&
((check.left+check.right+check.above+check.below)== 0 || check.color == 0 ) &&
(check.start+check.end+check.color+check.thickness)!=4){
        fprintf(stderr, "Input Error: not enough arguments for the function\n");
        exit(41);
    }
}

void get_options( int argc, char* argv[]){
    if (argc == 1) {
        fprintf(stderr, "Input Error: no arguments were entered\n");
        exit(41);
    }
    struct option long_options[] = {
        {"inverse", no_argument, NULL, 'n'},
        {"gray", no_argument, NULL, 'g'},
        {"resize", no_argument, NULL, 'r'},
        {"line", no_argument, NULL, 'l'},
        {"help", no_argument, NULL, 'h'},
        {"info", no_argument, NULL, 'I'},
        {"input", required_argument, NULL, 'i'},
        {"output", required_argument, NULL, 'o'},
        {"left_up", required_argument, NULL, 'u'},
        {"right_down", required_argument, NULL, 'd'},
        {"left", required_argument, NULL, 'L'},
        {"right", required_argument, NULL, 'R'},
        {"above", required_argument, NULL, 'A'},
        {"below", required_argument, NULL, 'B'},
        {"color", required_argument, NULL, 'C'},
        {"start", required_argument, NULL, 's'},
        {"end", required_argument, NULL, 'e'},
        {"thickness", required_argument, NULL, 't'},
        {NULL, 0, NULL, 0}
    };
    check_option check = do_check();
    int option;
    int left_x, up_y, right_x, down_y;
    int left = 0, right = 0, above = 0, below = 0;
    int start_x, start_y, end_x, end_y, thickness;
    char* input_file = (char*) calloc(sizeof(char), 128);
    char* output_file = (char*) calloc(sizeof(char), 128);
    Rgb color;
    while ((option = getopt_long(argc, argv, "ngrlhIi:o:u:d:L:R:A:B:C:s:e:t:",
long_options, NULL)) != -1) {
        switch (option) {
            case 'n':
                check.inverse++;
                break;
            case 'g':
```

```
                check.gray++;
                break;
            case 'r':
                check.resize++;
                break;
            case 'l':
                check.line++;
                break;
            case 'h':
                print_help();
                exit(0);
                break;
            case 'I':
                check.info++;
                break;
            case 'i':
                check.input++;
                strncpy(input_file, optarg, 127);
                break;
            case 'o':
                check.output++;
                strncpy(output_file, optarg, 127);
                break;
            case 'u':
                check.left_up++;
                if (sscanf(optarg, "%d.%d", &left_x, &up_y) != 2) {
                    fprintf(stderr, "Type Error: coordinates must be of the int
type\n");
                    exit(40);
                }
                break;
            case 'd':
                check.right_down++;
                if (sscanf(optarg, "%d.%d", &right_x, &down_y) != 2) {
                    fprintf(stderr, "Type Error: coordinates must be of the int
type\n");
                    exit(40);
                }
                break;
            case 'L':
                check.left++;
                if (sscanf(optarg, "%d", &left) != 1) {
                    fprintf(stderr, "Type Error: value of resize must be of the
int type\n");
                    exit(40);
                }
                break;
            case 'R':
                check.right++;
                if (sscanf(optarg, "%d", &right) != 1) {
                    fprintf(stderr, "Type Error: value of resize must be of the
int type\n");
                    exit(40);
                }
                break;
            case 'A':
                check.above++;
                if (sscanf(optarg, "%d", &above) != 1) {
                    fprintf(stderr, "Type Error: value of resize must be of the
int type\n");
                    exit(40);
                }
```

```
                break;
            case 'B':
                check.below++;
                if (sscanf(optarg, "%d", &below) != 1) {
                    fprintf(stderr, "Type Error: value of resize must be of the
int type\n");
                    exit(40);
                }
                break;
            case 'C':
                check.color++;
                int r, g, b;
                if (sscanf(optarg, "%d.%d.%d", &r, &g, &b) != 3) {
                    fprintf(stderr, "Type Error: color must be in the format
rrr.ggg.bbb\n");
                    exit(40);
                }
                color = check_color(r,g,b);
                break;
            case 's':
                check.start++;
                if (sscanf(optarg, "%d.%d", &start_x, &start_y) != 2) {
                    fprintf(stderr, "Type Error: coordinates must be of the int
type\n");
                    exit(40);
                }
                break;
            case 'e':
                check.end++;
                if (sscanf(optarg, "%d.%d", &end_x, &end_y) != 2) {
                    fprintf(stderr, "Type Error: coordinates must be of the int
type\n");
                    exit(40);
                }
                break;
            case 't':
                check.thickness++;
                if (sscanf(optarg, "%d", &thickness) != 1) {
                    fprintf(stderr, "Type Error: value of thinckness must be of
the int type\n");
                    exit(40);
                }
                if (thickness<0){
                    fprintf(stderr, "Value Error: value of thinckness must be
unsigned\n");
                    exit(42);
                }
                break;
            case '?':
                fprintf(stderr, "Input Error: wrong option\n");
                exit(41);
                break;
        }
    }
    if (check.output == 0) {
        strncpy(output_file, "out.bmp", 127);
    }
    if (check.input == 0){
        strncpy(input_file, argv[argc-1], 127);
    }
    if (strcmp(output_file,input_file)==0){
```

```
        fprintf(stderr, "Input Error: names of the input and output files should
not match\n");
        exit(41);
    }
    check_flags(check);
    BitmapFileHeader* bmfh =
(BitmapFileHeader*)malloc(sizeof(BitmapFileHeader));
    BitmapInfoHeader* bmif =
(BitmapInfoHeader*)malloc(sizeof(BitmapInfoHeader));
    check_bmp(input_file, bmfh, bmif);
    Rgb **arr = read_bmp(input_file, bmfh, bmif);
    int arr_height = bmif->height;
    int arr_width = bmif->width;
    if(check.info){
        print_info_header(*bmif);
        print_file_header(*bmfh);
    }
    if (check.inverse){
        check_coord(bmif, &left_x, &up_y, &right_x, &down_y);
        inverse(arr, *bmif, left_x, up_y, right_x, down_y);
        write_bmp(output_file, bmfh, bmif, arr);
    }
    if (check.gray){
        check_coord(bmif, &left_x, &up_y, &right_x, &down_y);
        gray(arr, *bmif, left_x, up_y, right_x, down_y);
        write_bmp(output_file, bmfh, bmif, arr);
    }
    if (check.resize){
        Rgb** new_arr = resize(arr, bmif, left, right, below, above, color);
        write_bmp(output_file, bmfh, bmif, new_arr);
        for (int i = 0; i < bmif->height; i++){
            free(new_arr[i]);
        }
        free(new_arr);
    }
    if (check.line){
        draw_line(arr, bmif, start_x, start_y, end_x, end_y, thickness, color);
        write_bmp(output_file, bmfh, bmif, arr);
    }
    for (int i = 0; i < arr_height; i++){
        free(arr[i]);
    }
    free(arr);
    free(bmif);
    free(bmfh);
}
```

Название файла: option.h

```
#pragma once

#include "help.h"
#include <string.h>
#include "bmp.h"
#include "drawing.h"
#include "struct.h"
#include <getopt.h>

check_option do_check();
void check_flags(check_option check);
void get_options( int argc, char* argv[]);
```

Название файла: struct.h

```
#pragma once

#pragma pack(push,1)

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;

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

#pragma pack(pop)

typedef struct {
    int inverse;
    int gray;
    int resize;
    int line;
    int input;
    int output;
    int left_up;
    int right_down;
    int left;
    int right;
    int above;
    int below;
    int color;
    int start;
    int end;
    int thickness;
    int info;
} check_option;
```

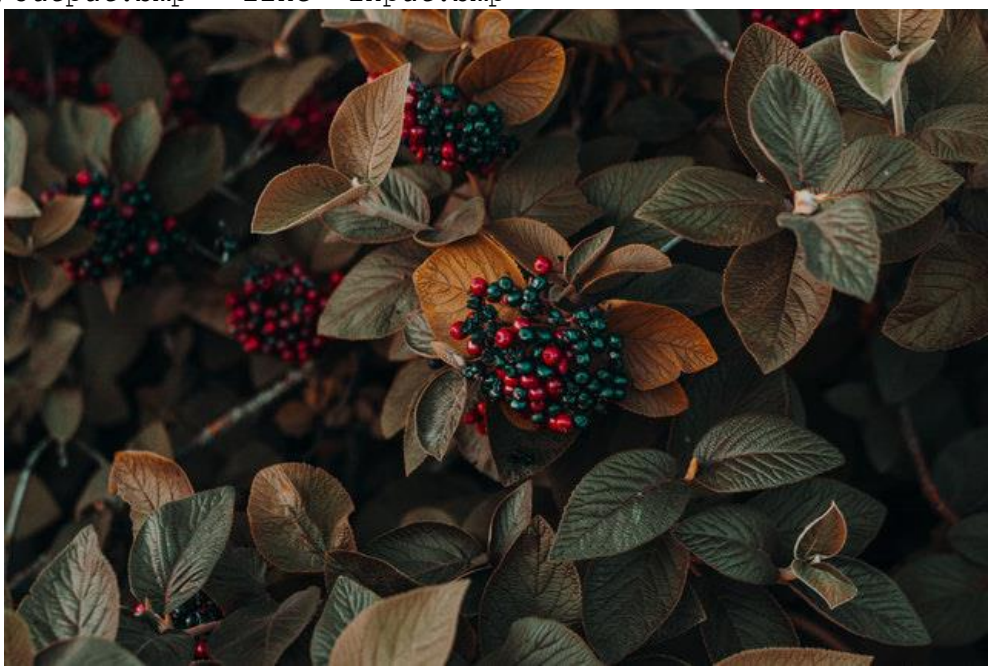
ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ

Таблица 1 – Результаты тестирования обработки ошибок

№ п/п	Входные данные	Выходные данные	Комментарии
1.	./cw --inverse --gray --left_up 10.100 --right_down 1000.1000 input.bmp	Input Error: impossible use several functions at once	Попытка использовать две функции сразу
2.	./cw --resize --color 265.0.240 --left 100 --above -100 --below 30 --right -20	Value Error: component values must be in the range 0 to 255	Неправильное значение
3.	./cw -inverse --left_up 100.100 --right_down 1000.1000 out.bmp	Input Error: names of the input and output files should not match	Ошибка, так как имя выходного файла по умолчанию совпадает с введенным
4.	./cw --resize --color 255.0.240 input.bmp	Input Error: not enough arguments for the function	Нехватка аргументов
5.	./cw -inverse --left_up 100.100 --right_down 1000.1000 --right_down 1000.1000 input.bmp	Input Error: forbidden use the same flag multiple times	Два раза вызван один и тот же флаг
6.	./cw --left_up 10.100 --right_down 1000.1000 input.bmp	Input Error: none of the action functions were called	Не была вызвана одна из основных функций
7.	./cw --color 251.101.221 --line --end -529.9 --thickness -10 --start 474.1605 --output ./output.bmp input.bmp	Value Error: value of thinckness must be unsigned	Введена отрицательная толщина
8.	./cw --inverse --left_up sd.100 --right_down 1000.1000 input.bmp	Type Error: coordinates must be of the int type	Координаты должны быть числами

Фото для обработки – рисование линии

```
./cw --end 20.40 --thickness 30 --start 401.300  --color 245.64.44 --
output ./output.bmp --line  input.bmp
```



Результат работы программы:

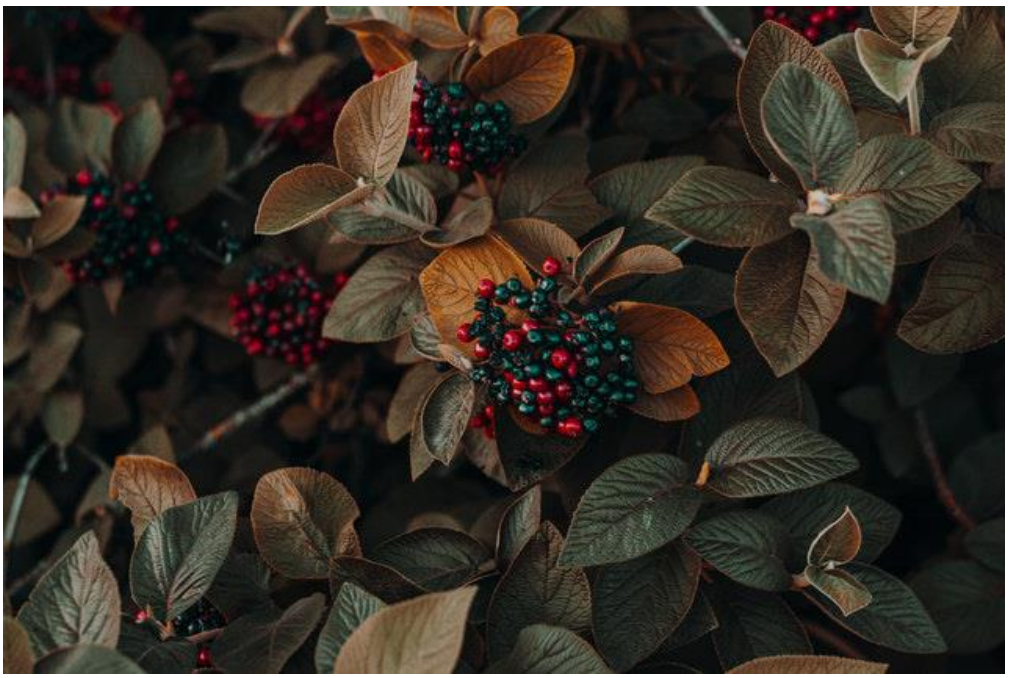
Фото для обработки – инвертирование заданной области

```
./cw --inverse --left_up 100.100 --right_down 1000.1000 input.bmp
```

Результат работы программы:

Фото для обработки – преобразование в ч/б изображение заданной области

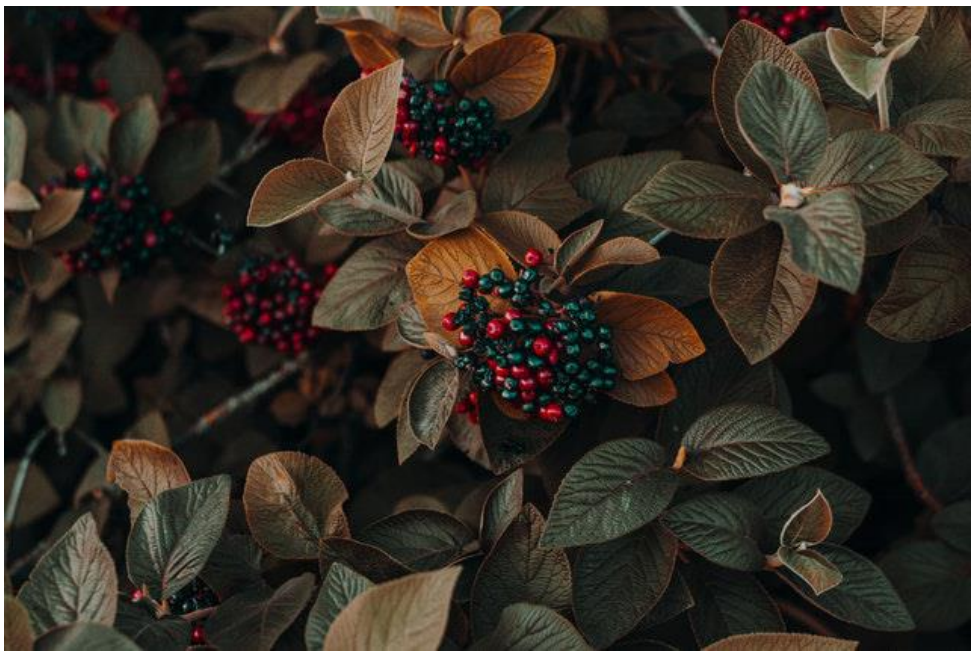
```
./cw --gray --left_up 100.100 --right_down 600.400 input.bmp
```



Результат работы программы:

Фото для обработки – преобразование в ч/б изображение заданной области

```
./cw --gray --left_up 100.100 --right_down 600.400 input.bmp
```



Результат работы программы: