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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка BMP изображения.

Студентка гр. 3343 _____ Ермолаева В. А.

Преподаватель _____ Государкин Я. С.

Санкт-Петербург

2024

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студентка Ермолаева В. А.

Группа 3343

Тема работы: Обработка BMP изображения.

Исходные данные:
Программа обязательно должна иметь CLI (опционально дополнительное использование GUI).
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).

Содержание пояснительной записки:

- Аннотация
- Содержание
- Введение
- Описание задачи и требований
- Описание структур
- Описание реализованных функций
- Тестирование
- Заключение
- Приложение А. Исходный код программы

Предполагаемый объем пояснительной записки:

Не менее 35 страниц.

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 29.05.2024

Дата защиты реферата: 29.05.2024

Студентка _____ Ермолаева В. А.

Преподаватель _____ Государкин Я. С.

АННОТАЦИЯ

Основным поставленным заданием курсовой работы является разработка программы, направленной на обработку BMP изображения, поданного на вход пользователем. Производится реализация нескольких структур для хранения информации об изображении и его дальнейшей обработки. В программу заложено три базовые функции, включающие в себя рисование квадрата с диагоналями, фильтр rgb-компонент и поворот изображения или его части на 90/180/270 градусов. Помимо этого пользователю доступны дополнительные функции, выводящие информацию о программе или поданном на вход изображении. Взаимодействие пользователя с программой осуществляется при помощи интерфейса командной строки.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью работы является разработка программы на языке Си, направленной на обработку изображений формата BMP (от англ. Bitmap Picture). В основные задачи входят изучение и практическое применение навыков, необходимых для:

- реализации структур, необходимых для хранения и обработки изображений
- успешного применения функционала библиотек языка Си для работы с изображениями к практическим задачам
- обработке ошибок, которые могут возникнуть в процессе исполнения программы
- реализации методов обработки изображений согласно поставленным задачам.

Среди задач данной курсовой работы имеется реализация интерфейса командной строки с использованием getopt, своевременное уведомление пользователя о некорректно введенных данных, а также обеспечение работоспособности программы на неожиданных данных, введенных пользователем.

ОПИСАНИЕ ЗАДАЧИ И ТРЕБОВАНИЙ

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`

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

ОПИСАНИЕ СТРУКТУР

Программа включает в себя пять структур, необходимых для обработки изображений.

- BitmapFileHeader: описывает заголовок BMP файла. Она используется для идентификации файла как BMP, определения его размера и указания смещения, где начинается массив пикселей изображения.
- BitmapInfoHeader: храненит информацию об изображении. Содержит сведения о длине изображения, его ширине, глубине цвета, количестве цветов в цветовой таблице и т. д.
- RGB: хранит информацию об интенсивности красной, синей и зеленой компонент пикселя.
- Error: используется для более удобной обработки ошибок, хранит значение, указывающее на наличие ошибки и сообщение, которое следует вывести при ее возникновении.
- Options: хранит считанные значения аргументов, поданных в программу пользователем.

ОПИСАНИЕ РЕАЛИЗОВАННЫХ ФУНКЦИЙ

Программа состоит из функций, описание которых представлено ниже:

- int main(): главная функция программы, выводит базовую информацию о курсовой работе и вызывает функцию обработки аргументов, введенных пользователем.
- void process_args(int argc, char* argv[]): обрабатывает введенные пользователем аргументы, проводит предварительную обработку ошибок и при их отсутствии вызывает функцию обработки изображения.
- Error process_image(char* filename, Options opts): функция, которая производит считывание изображения, его преобразование и запись нового файла. На этом этапе проверяется большая часть ошибок. Возвращает структуру ошибки, хранящую информацию о том, возникли ли в процессе изменения изображения ошибки, и если да, то какие.
- int check_options(Options* opts, Error* err): проверяет аргументы, введенные пользователем на корректность.
- int check_int(char* str): проверяет, является ли переданная строка целым числом.
- void rotate(RGB** rgb, int height, int width, int x0, int y0, int x1, int y1, int angle): поворачивает изображение (часть) на 90/180/270 градусов. В случае, если повернутая часть изображения выходит за границы изображения, рисуется только вместившаяся часть.
- void swap_channels(RGB** rgb, int height, int width, char* component_name, unsigned int component_value): применяет к изображению фильтр rgb-компонент.
- void draw_square(RGB** rgb, BitmapInfoHeader* info, int x, int y, int side_size, int thickness, char* colour, int fill, char* fill_colour): рисует по указанным координатам квадрат, который может быть залит или нет. В случае, если квадрат выходит за границы изображения, отрисовывается только та часть квадрата, которая на него попадает.

- void get_coordinates(char* left_up, int* x, int* y): записывает в переданные адреса переменных извлеченные координаты.
- RGB get_colour(char* colour): возвращает пиксель, содержащий цветовые компоненты, извлеченные из переданной строки.
- int check_coordinate_format(char* coord): проверяет переданную строку с координатами на корректность.
- int check_color(char* color): проверяет переданную строку с цветом на корректность.
- void line(RGB** rgb, BitmapInfoHeader* info, int x0, int y0, int x1, int y1, int thickness, RGB pixel): рисует на изображении линию по алгоритму Брезенхэма.
- void circle(RGB** rgb, BitmapInfoHeader* info, int radius, int x1, int y1, RGB pixel): рисует на изображении окружность по алгоритму Брезенхэма.
- int check_coordinates(int x, int y, int height, int width): проверяет координаты на то, принадлежат ли они изображению.
- void paint_pixel(RGB* pixel, int b, int g, int r): записывает в пиксель указанные цветовые компоненты.
- void write_bmp(char* file_name, RGB** rgb, BitmapFileHeader* file, BitmapInfoHeader* info): записывает в файл изображение.
- RGB** read_bmp(char* file_name, BitmapFileHeader* file, BitmapInfoHeader* info, Error* err): считывает изображение из файла.
- void check_bmp(BitmapFileHeader* file, Error* err): проверяет файл с изображением на соответствие BMP формату.
- void delete_RGB(RGB** rgb, int height): очищает память, выделенную под хранение изображения.
- RGB** create_RGB(int height, int width): выделяет память под хранение изображения.
- void initialize_opts(Options* opts): устанавливает в опции значения по умолчанию.

- void print_info_header(BitmapInfoHeader* header): выводит значения всех полей структуры BitmapInfoHeader, т. е. подробную информацию об изображении.
- void print_file_header(BitmapFileHeader* header): выводит значения всех полей структуры BitmapFileHeader, т. е. информацию о файле.
- void print_help(): выводит подробную информацию о программе, ее флагах и принимаемых значениях.
- char* strdup (const char* s): копирует строку.

Более подробно с функциями и их содержанием можно ознакомиться в приложении А.

ТЕСТИРОВАНИЕ

Исходное изображение:

Рис. 1: Исходное изображение

1. Тестирование функции squared_lines.

   Аргументы для запуска: --fill_color 56.98.217 --thickness 11 --side_size 362 --color 110.35.109 --output ./output.bmp --fill --input ./food.bmp --left_up 199.347 —squared_lines

Рис. 2: Тестирование функции squared_lines

2. Тестирование функции rgbfilter.

   Аргументы для запуска: --component_value 69 --output ./output.bmp --rgbfilter --input ./food.bmp --component_name "green"

Рис. 3: Результат работы функции rgbfilter

3. Тестирование функции rgbfilter.

   Аргументы для запуска: --angle 90 --left_up 200.100 --right_down 400.400 --input ./food.bmp --output ./output.bmp —rotate

Рис. 4: Результат работы функции rotate

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были изучены и применены на практике знания о формате BMP, были освоены навыки работы с изображениями формата BMP на языке Си, их изменением, созданием и считыванием. Были обработаны неожиданный ввод пользователя, потенциальные ошибки, которые могут возникнуть в процессе выполнения программы, а также реализован интерфейс командной строки с использованием getopt.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

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

typedef struct {
    int value;
    char* message;
} Error;

typedef struct {
    int rgb_filter_flag;
    int squared_lines_flag;
    int fill_flag;

    char* output_name;
    char* input_name;

    int rotate_flag;
    int info_flag;

    char* component_name;
    char* component_value;

    char* left_up;
    char* right_down;

    char* side_size;
    char* thickness;
    char* angle;

    char* color;
    char* fill_color;
} Options;
```

```
#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 coloursIncolourTable;
    unsigned int importantcolourCount;
} BitmapInfoHeader;

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

#pragma pack(pop)

char* strdup (const char* s) {
    size_t slen = strlen(s);
    char* result = malloc(slen + 1);

    if(result == NULL) return NULL;
    memcpy(result, s, slen + 1);
    return result;
}
```

```
void print_help() {
    printf("Флаги, используемые в программе:\n"
        "--help: Выводит справочную информацию.\n"
        "--input: Задаёт имя входного изображения.\n"
        "--output: Задаёт имя выходного изображения.\n"
        "--info: Вывод информации об изображении.\n"
        "--squared_lines: Рисование квадрата с диагоналями.\n"
        "--side_size: Размер стороны. Принимает число больше 0.\n"
        "--thickness: Толщина линии. Принимает число больше 0.\n"
           "--color: Цвет линии, задаётся строкой \'rrr.ggg.bbb\', где rrr/ggg/bbb – числа, задающие цветовую компоненту.\n"
          "--fill: Может быть залит или нет. Флаг работает как бинарное значение: флага нет – false , флаг есть – true.\n"
        "--fill_color: Цветом которым он залит, если пользователем выбран залитый. работает аналогично флагу \'--color\'.\n"
        "--rgbfilter: Фильтр rgb-компонент.\n"
        "--component_name: Какую компоненту требуется изменить. Возможные значения \'red\', \'green\' и \'blue\'\n"
          "--component_value: В какое значение ее требуется изменить. Принимает значение в виде числа от 0 до 255.\n"
        "--rotate: Поворот изображения (части) на 90/180/270 градусов.\n"
          "--left_up: Координаты левого верхнего угла области. Значение задаётся в формате \'left.up\', где left – координата по x, up – координата по y.\n"
         "--right_down: Координаты правого нижнего угла области. Значение задаётся в формате \'right.down\', где right – координата по x, down – координата по y.\n"
                                "--angle: Угол поворота. Возможные значения: \'90\', \'180\', \'270\'.\n");
}

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("coloursIncolourTable:\t%x     (%u)\n",     header-
>coloursIncolourTable, header->coloursIncolourTable);
                 printf("importantcolourCount:\t%x     (%u)\n",     header-
>importantcolourCount, header->importantcolourCount);
}

void initialize_opts(Options* opts) {
    opts->rgb_filter_flag = 0;
    opts->squared_lines_flag = 0;

    opts->fill_flag = 0;
    opts->rotate_flag = 0;
    opts->info_flag = 0;

    opts->input_name = "";
    opts->output_name = "out.bmp";
    opts->fill_color = NULL;
```

```
    opts->component_name = "";
    opts->component_value = NULL;
    opts->left_up = NULL;

    opts->right_down = NULL;
    opts->side_size = NULL;
    opts->thickness = NULL;

    opts->angle = NULL;
    opts->color = NULL;
    opts->fill_color = NULL;
}

RGB** create_RGB(int height, int width) {
    RGB** rgb = malloc(height * sizeof(RGB *));

    for(int i = 0; i < height; i++)
        rgb[i] = malloc(width * sizeof(RGB) + ((4 - (width * sizeof(RGB))
% 4) % 4));

    return rgb;
}

void delete_RGB(RGB** rgb, int height) {
    for(int i = 0; i < height; i++)
        free(rgb[i]);

    free(rgb);
}

void check_bmp(BitmapFileHeader* file, Error* err) {
    if (file->signature != 0x4d42 && file->signature != 0x424d) {
        err->value = 1;
        err->message = "Given file is not a BMP.";
    }
}
```

```
RGB** read_bmp(char* file_name, BitmapFileHeader* file, BitmapInfoHeader*
info, Error* err) {
    FILE *f = fopen(file_name, "rb");
    if (!f) {
        err->value = 1;
        err->message = "Couldn't open given file.";
        return create_RGB(0, 0);
    }

    fread(file, 1, sizeof(BitmapFileHeader), f);
    fread(info, 1, sizeof(BitmapInfoHeader), f);

    check_bmp(file, err);
    if (err->value) {
        fclose(f);
        return create_RGB(0, 0);
    }

    fseek(f, file->pixelArrOffset, SEEK_SET);

    int height = info->height;
    int width = info->width;

    RGB** rgb = create_RGB(height, width);
    for(int i = 0; i < height; i++)
          fread(rgb[i], 1, width * sizeof(RGB) + ((4 - (info->width *
sizeof(RGB)) % 4) % 4), f);

    fclose(f);
    return rgb;
}

void write_bmp(char* file_name, RGB** rgb, BitmapFileHeader* file,
BitmapInfoHeader* info) {
    FILE* f = fopen(file_name, "wb");

    fwrite(file, 1, sizeof(BitmapFileHeader), f);
    fwrite(info, 1, sizeof(BitmapInfoHeader), f);
```

```
    fseek(f, file->pixelArrOffset, SEEK_SET);

    for(int i = 0; i < info->height; i++)
        fwrite(rgb[i], 1, info->width * sizeof(RGB) + ((4 - (info->width
* sizeof(RGB)) % 4) % 4), f);

    fclose(f);
}

void paint_pixel(RGB* pixel, int b, int g, int r) {
    pixel->b = b;
    pixel->g = g;
    pixel->r = r;
}

int check_coordinates(int x, int y, int height, int width) {
    if (x >= 0 && y >= 0 && x < width && y < height) return 1;
    return 0;
}

void circle(RGB** rgb, BitmapInfoHeader* info, int radius, int x1, int
y1, RGB pixel) {
    int x = 0;
    int y = radius;
    int delta = 1 - 2 * radius;
    int error = 0;

    while (y >= x) {
        for (int i = x1 - x; i <= x1 + x; i++) {
            if (check_coordinates(i, y1+y, info->height, info->width))
rgb[y1 + y][i] = pixel;
                if (check_coordinates(i, y1-y, info->height, info-
>width))rgb[y1 - y][i] = pixel;
        }

        for (int i = x1 - y; i <= x1 + y; i++) {
                if (check_coordinates(i, y1+x, info->height, info-
>width))rgb[y1 + x][i] = pixel;
```

```
                    if (check_coordinates(i, y1-x, info->height, info-
>width))rgb[y1 - x][i] = pixel;
        }

        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 line(RGB** rgb, BitmapInfoHeader* info, int x0, int y0, int x1, int
y1, int thickness, RGB pixel) {
    int dx = abs(x1 - x0);
    int dy = abs(y1 - y0);

    int sx = x0 < x1 ? 1 : -1;
    int sy = y0 < y1 ? 1 : -1;

    int err = (dx > dy ? dx : -dy) / 2;

    while(1) {
        circle(rgb, info, (thickness + 1)/2, x0, y0, pixel);

        if (x0 == x1 && y0 == y1)
            break;

        if (err > -dx) {
            err -= dy;
            x0 += sx;
        }

        if (err < dy) {
```

```
            err += dx;
            y0 += sy;
        }
    }
}

int check_color(char* color) {
    int dot_count = 0;
    int num = 0;

    for (int i = 0; i < strlen(color); i++) {
        if (!isdigit(color[i]) && color[i] != '.') return 0;
        if (color[i] == '.') dot_count++;
    }

    if (dot_count != 2)
        return 0;

    if (color[0] == '.' || color[strlen(color) - 1] == '.')
        return 0;

    char* token = strtok(strdup(color), ".");
    for (int i = 0; i < 3; i++) {
        if (token == NULL || !isdigit(token[0])) return 0;
        num = atoi(token);

        if (num < 0 || num > 255) return 0;
        token = strtok(NULL, ".");
    }

    return 1;
}

int check_coordinate_format(char* coord) {
    int dot_count = 0;

    for (int i = 0; i < strlen(coord); i++) {
        if (!isdigit(coord[i]) && coord[i] != '.') return 0;
        if (coord[i] == '.') dot_count++;
```

```
    }

    if (dot_count != 1)
        return 0;

    if (coord[0] == '.' || coord[strlen(coord) - 1] == '.')
        return 0;

    return 1;
}

RGB get_colour(char* colour) {
    RGB pixel;
    char* str = strdup(colour);
    char* end;

    int red = strtol(str, &end, 10);
    int green = strtol(end + 1, &end, 10);
    int blue = strtol(end + 1, NULL, 10);

    paint_pixel(&pixel, blue, green, red);
    return pixel;
}

void get_coordinates(char* left_up, int* x, int* y) {
    char* str = strdup(left_up);
    char* end;

    *x = strtol(str, &end, 10);
    *y = strtol(end + 1, NULL, 10);
}

void draw_square(RGB** rgb, BitmapInfoHeader* info, int x, int y, int
side_size, int thickness, char* colour, int fill, char* fill_colour) {
    RGB pixel = get_colour(colour);

    if (fill) {
        RGB pixel_filled = get_colour(fill_colour);
```

```
        for (int i = y+thickness/2; i < y+side_size-thickness/2; i++) {
            for (int j = x+thickness/2; j < x+side_size-thickness/2; j++)
{
                 if (check_coordinates(j, i, info->height, info->width))
rgb[i][j] = pixel_filled;
            }
        }

    }

    line(rgb, info, x, y, x + side_size, y, thickness, pixel);
    line(rgb, info, x, y, x, y + side_size, thickness, pixel);
      line(rgb, info, x, y + side_size, x + side_size, y + side_size,
thickness, pixel);
      line(rgb, info, x + side_size, y, x + side_size, y + side_size,
thickness, pixel);

      line(rgb, info, x, y, x + side_size, y + side_size, thickness,
pixel);
      line(rgb, info, x + side_size, y, x, y + side_size, thickness,
pixel);
}

void swap_channels(RGB** rgb, int height, int width, char*
component_name, unsigned int component_value) {
    for(int i = 0; i < height; i++) {
        for(int j = 0; j < width; j++) {
            if (!strcmp(component_name, "green"))
                rgb[i][j].g = component_value;
            else if (!strcmp(component_name, "blue"))
                rgb[i][j].b = component_value;
            else
                rgb[i][j].r = component_value;
        }
    }
}

void rotate(RGB** rgb, int height, int width, int x0, int y0, int x1, int
y1, int angle) {
```

```
    RGB** rgb_rotated = create_RGB(height, width);

    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            rgb_rotated[i][j] = rgb[i][j];
        }
    }

    int xc = (x1 + x0) / 2; int yc = (y1 + y0) / 2;

    int x; int y;
    int xr; int yr;

    for (int i = y0; i < y1; i++) {
        for (int j = x0; j < x1; j++) {
            switch(angle) {
                case 90:
                    xr = j - xc; yr = i - yc;
                    x = xc - yr; y = yc + xr;

                    if (!((y1 + y0) % 2 == 0 && (x1 + x0) % 2 == 0)) {
                        if ((y1 + y0) % 2 == 0) y -= 1;
                        if ((y1 + y0) % 2 != 0) y += 1;
                    }

                    if ((x1 + x0) % 2 != 0) x -= 1;

                          if (check_coordinates(j, i, height, width) &&
check_coordinates(x, y, height, width))
                        rgb[y][x] = rgb_rotated[i][j];
                    break;
                case 180:
                    x = x1 - j - 1 + x0; y = y1 - i - 1 + y0;

                          if (check_coordinates(j, i, height, width) &&
check_coordinates(x, y, height, width))
                        rgb[i][j] = rgb_rotated[y][x];
                    break;
                case 270:
```

```
                    xr = j - xc; yr = i - yc;
                    x = xc + yr; y = yc - xr;

                    if ((y1 + y0) % 2 == 0) y -= 1;

                        if (check_coordinates(j, i, height, width) &&
check_coordinates(x, y, height, width))
                        rgb[y][x] = rgb_rotated[i][j];
                    break;
                }
            }
        }

    delete_RGB(rgb_rotated, height);
}

int check_int(char* str) {
    if (!str) return 0;

    while (*str) {
        if (*str < '0' || *str > '9') return 0;
        str++;
    }

    return 1;
}

int check_options(Options* opts, Error* err) {
    if (opts->rgb_filter_flag) {
        if (!opts->component_name || !opts->component_value) {
            err->value = 1;
            err->message = "Some required flags are missing.";
            return 1;
        }

             if (!opts->component_name || (strcmp(opts->component_name,
"green") && strcmp(opts->component_name, "blue") && strcmp(opts-
>component_name, "red"))) {
            err->value = 1;
```

```
            err->message = "Incorrect component name, must be blue, red
or green.";
            return 1;
        }

        if (!check_int(opts->component_value)) {
            err->value = 1;
                 err->message = "Incorrect component value, must be an
integer.";
            return 1;
        }

                if (atoi(opts->component_value) < 0 || atoi(opts-
>component_value) > 255) {
            err->value = 1;
             err->message = "Incorrect component value, must be between 0
and 255.";
            return 1;
        }
    }

    if (opts->squared_lines_flag) {
         if (!opts->left_up || !opts->side_size || !opts->thickness || !
opts->color || ((opts->fill_flag && !opts->fill_color))) {
            err->value = 1;
            err->message = "Some required flags are missing.";
            return 1;
        }

        if(!check_coordinate_format(opts->left_up)) {
            err->value = 1;
                       err->message = "Given coordinates must be in
format \"x.y\".";
            return 1;
        }

        if (!check_int(opts->side_size) || !check_int(opts->thickness)) {
            err->value = 1;
```

```
            err->message = "Incorrect side_size or thickness values, must
be integers.";
            return 1;
        }

        if (atoi(opts->side_size) < 0 || atoi(opts->thickness) < 0) {
            err->value = 1;
            err->message = "Numeric values must be positive.";
            return 1;
        }

               if (!check_color(opts->color) || (opts->fill_flag && !
check_color(opts->fill_color))) {
            err->value = 1;
            err->message = "Color must be given in format \"0.0.0\".";
            return 1;
        }
    }

    if (opts->rotate_flag) {
        if(!opts->left_up || !opts->right_down || !opts->angle) {
            err->value = 1;
            err->message = "Some required flags are missing.";
            return 1;
        }

                        if(!check_coordinate_format(opts->left_up) || !
check_coordinate_format(opts->right_down)) {
            err->value = 1;
                         err->message = "Given coordinates must be in
format \"x.y\".";
            return 1;
        }

        if (!check_int(opts->angle)) {
            err->value = 1;
            err->message = "Incorrect angle value, must be an integer.";
            return 1;
        }
```

```
          if (atoi(opts->angle) % 90 != 0 || atoi(opts->angle) < 0 ||
atoi(opts->angle) > 270) {
            err->value = 1;
                 err->message = "Rotation angle must be 90, 180 or 270
degrees.";
            return 1;
        }
    }

    return 0;
}

Error process_image(char* filename, Options opts) {
    BitmapFileHeader file;
     BitmapInfoHeader info;

    Error err;
    err.value = 0;

    RGB** rgb = read_bmp(filename, &file, &info, &err);
    if (err.value == 1) return err;

    if (opts.info_flag) {
        print_file_header(&file);
        print_info_header(&info);
    }

    if (opts.rgb_filter_flag) {
         if(!check_options(&opts, &err)) swap_channels(rgb, info.height,
info.width, opts.component_name, atoi(opts.component_value));
    }

    if (opts.squared_lines_flag) {
        if (!check_options(&opts, &err)) {
            int x; int y;
            get_coordinates(opts.left_up, &x, &y);

            if (opts.fill_flag)
```

```
                        draw_square(rgb, &info, x, info.height - y -
atoi(opts.side_size),     atoi(opts.side_size),     atoi(opts.thickness),
opts.color, 1, opts.fill_color);
            else
                        draw_square(rgb, &info, x, info.height - y -
atoi(opts.side_size),     atoi(opts.side_size),     atoi(opts.thickness),
opts.color, 0, NULL);
        }
    }

    if (opts.rotate_flag) {
        if(!check_options(&opts, &err)) {
            int x0, y0, x1, y1;

            get_coordinates(opts.left_up, &x0, &y0);
            get_coordinates(opts.right_down, &x1, &y1);

            int side_size = abs(y1 - y0);

            if (y1 < y0 || x1 < x0) {
                int temp_x = x0; int temp_y = y0;
                x0 = x1; y0 = y1;
                x1 = temp_x; y1 = temp_y;
            }

             rotate(rgb, info.height, info.width, x0, info.height - y0 -
side_size, x1, info.height - y1 + side_size, atoi(opts.angle));
        }
    }

    write_bmp(opts.output_name, rgb, &file, &info);

    delete_RGB(rgb, info.height);
    return err;
}

int check_file(char* filename) {
    BitmapFileHeader file;
    BitmapInfoHeader info;
```

```
    Error err;
    err.value = 0;

    RGB** rgb = read_bmp(filename, &file, &info, &err);

    if (err.value == 0) {
        delete_RGB(rgb, info.height);
        return 1;
    }

    delete_RGB(rgb, info.height);
    return 0;
}

void process_args(int argc, char* argv[]) {
    struct option long_options[] = {
        {"help", no_argument, 0, 'h'},
        {"output", required_argument, 0, 'o'},
        {"info", no_argument, 0, 5},
        {"input", required_argument, 0, 'i'},
        {"rgbfilter", no_argument, 0, 1},
        {"component_name", required_argument, 0, 'n'},
        {"component_value", required_argument, 0, 'v'},
        {"squared_lines", no_argument, 0, 2},
        {"left_up", required_argument, 0, 'l'},
        {"side_size", required_argument, 0, 's'},
        {"thickness", required_argument, 0, 't'},
        {"color", required_argument, 0, 'c'},
        {"fill", no_argument, 0, 3},
        {"fill_color", required_argument, 0, 'f'},
        {"rotate", no_argument, 0, 4},
        {"right_down", required_argument, 0, 'r'},
        {"angle", required_argument, 0, 'a'},
        {0, 0, 0, 0}
    };

    char* extra_args[argc];
    int extra_args_count = 0;
```

```
    Options opts;
    initialize_opts(&opts);

    int opt;
    opterr = 0;

      while ((opt = getopt_long(argc, argv, "hi:o:n:v:l:s:t:c:f:r:a:",
long_options, NULL)) != -1) {
        switch (opt) {
            case 1:
                opts.rgb_filter_flag = 1;
                break;
            case 2:
                opts.squared_lines_flag = 1;
                break;
            case 3:
                opts.fill_flag = 1;
                break;
            case 4:
                opts.rotate_flag = 1;
                break;
            case 5:
                opts.info_flag = 1;
                break;
            case 'n':
                opts.component_name = optarg;
                break;
            case 'v':
                opts.component_value = optarg;
                break;
            case 'l':
                opts.left_up = optarg;
                break;
            case 's':
                opts.side_size = optarg;
                break;
            case 't':
                opts.thickness = optarg;
```

```
            break;
        case 'c':
            opts.color = optarg;
            break;
        case 'f':
            opts.fill_color = optarg;
            break;
        case 'r':
            opts.right_down = optarg;
            break;
        case 'a':
            opts.angle = optarg;
            break;
        case 'h':
            print_help();
            break;
        case 'o':
            opts.output_name = optarg;
            break;
        case 'i':
            opts.input_name = optarg;
            break;
        case '?':
            extra_args[extra_args_count++] = argv[optind - 1];
            break;
    }
}

while (optind < argc) {
    extra_args[extra_args_count++] = argv[optind++];
}

Error err;
err.value = 0;

if (optind == 1) print_help();

if (extra_args_count > 0) {
    for (int i = 0; i < extra_args_count; i++) {
```

```
            if (check_file(extra_args[i])) {
                opts.input_name = extra_args[i];
            }

                else printf("Extra argument \'%s\' will be ignored.\n",
extra_args[i]);
        }
    }

    if (strstr(opts.input_name, opts.output_name)) {
        err.value = 1;
        err.message = "Can't use the same input and output file names.";
    }

    if (!err.value) err = process_image(opts.input_name, opts);
    if (err.value) printf("%s\n", err.message);
}

int main(int argc, char* argv[]) {
    printf("Course work for option 4.12, created by Vera Ermolaeva.\n");
    process_args(argc, argv);
    return 0;
}
```