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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка BMP изображения

Студент гр. 3343 _____ Синицкая Д.В.

Преподаватель _____ Государкин Я.С.

Санкт-Петербург

2024

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Условия задания (Вариант 4.1):

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).

Программа должна иметь следующую функции по обработке изображений:

1. Рисование окружности. Флаг для выполнения данной операции: `--circle`. Окружность определяется:
    1. координатами ее центра и радиусом. Флаги `--center` и `--radius`. Значение флаг `--center` задаётся в формате `x.y`, где x – координата по оси x, y – координата по оси y. Флаг `--radius` На вход принимает число больше 0Количество частей по “оси” X. Флаг `--number_y`. На вход принимает число больше 1.
    2. толщиной линии окружности. Флаг `--thickness`. На вход принимает число больше 0
    3. цветом линии окружности. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)

4. окружность может быть залитой или нет. Флаг `--fill`. Работает как бинарное значение: флага нет – false , флаг есть – true.
5. цветом которым залита сама окружность, если пользователем выбрана залитая окружность. Флаг `--fill_color` (работает аналогично флагу `--color`)

2. Отражение заданной области. Флаг для выполнения данной операции: `--mirror`. Этот функционал определяется:
   1. выбором оси относительно которой отражать (горизонтальная или вертикальная). Флаг `--axis`, возможные значения `x` и `y`
   2. Координатами левого верхнего угла области. Флаг `--left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y
   3. Координатами правого нижнего угла области. Флаг `--right_down`, значение задаётся в формате `right.down`, где right – координата по x, down – координата по y
3. Копирование заданной области. Флаг для выполнения данной операции: `--copy`. Функционал определяется:
   1. Координатами левого верхнего угла области-источника. Флаг `--left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y
   2. Координатами правого нижнего угла области-источника. Флаг `--right_down`, значение задаётся в формате `right.down`, где right – координата по x, down – координата по y
   3. Координатами левого верхнего угла области-назначения. Флаг `--dest_left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 29.05.2024

Дата защиты реферата: 29.05.2024

АННОТАЦИЯ

В ходе курсовой работы реализована программа, осуществляющая обработку BMP изображения. Для взаимодействия с программой реализован интерфейс командной строки (CLI). Программа реализует следующие функции: отрисовка окружности в двух вариантах с заливкой или без, отражение заданной области в двух вариантах относительно оси X или относительно оси У, копирование заданной области.

ВВЕДЕНИЕ

Цель работы: понять структуру BMP изображения, научиться работать с BMP изображением на языке программирования C, реализовать программу, исполняющую несколько функций по обработке изображения, его считывание и запись, взаимодействие с которой должно осуществляться с помощью интерфейса командной строки.

1. ОПИСАНИЕ СТРУКТУРЫ ПРОГРАММЫ

Описание структур:

1. *BitmapFileHeader* – структура для хранения заголовка BMP файла.
2. *BitmapInfoHeader* – структура для хранения информационного заголовка, содержащего информацию о параметрах BPM файла.
3. *Rgb* – структура для хранения цветовой компоненты пикселя.
4. *Options* – структура для хранения информации об аргументах и состоянии флагов, используемых для взаимодействия с программой.
5. *Errors* – структура для хранения информации об ошибках.

Описание функций и методов:

1. *void reference()* – метод вывода информации из справки на экран.
2. *void print_file_header(BitmapFileHeader *header)* – метод вывода информации о заголовке BMP файла на экран. Принимает на вход указатель на структуру заголовка BMP файла.
3. *void print_info_header(BitmapInfoHeader *header)* – метода вывода информации о информационном заголовке BMP файла на экран. Принимает на вход указатель на структуру информационного заголовка BMP файла.
4. *void write_bmp(char* file_name, Rgb **arr, BitmapFileHeader *bmfh, BitmapInfoHeader *bmif, Errors* err)* – метод записи данных в BMP файл. Принимает на вход имя файла, двойной указатель на массив пикселей, указатель на структуру заголовка BMP файла, указатель на структуру информационного заголовка BMP файла и указатель на структуру ошибок.
5. *Rgb **read_bmp(char* file_name, BitmapFileHeader *bmfh, BitmapInfoHeader *bmif, Errors *err)* – функция чтения файла. Принимает на вход имя файла, указатель на структуру заголовка BMP файла, указатель на структуру информационного заголовка BMP файла и указатель на структуру ошибок. Возвращает двойной указатель на массив пикселей.

6. *void freeRgbArr(Rgb** arr, unsigned int H)* – метод очистки памяти, выделенной под массив пикселей. Принимает на вход двойной указатель на массив пикселей и высоту изображения.
7. *void coordinate_conversion(int* mas, char* coordinate)* – метод преобразования координат от строчного типа к числовому. Принимает на вход указатель на массив и строку с координатами.
8. *void check_parametres(Options* option, BitmapInfoHeader* bmif, Errors *err)* – метод проверки параметров флагов на ошибки. Принимает на вход указатель на структуру параметров, указатель на структуру информационного заголовка BMP файла и указатель на структуру ошибок.
9. *void xLine(Rgb** rgb, int x1, int x2, int y, Rgb color, BitmapInfoHeader* bmif)* – метод рисования горизонтальной линии пикселей заданного цвета между внутренним и внешним радиусом по координате y. Принимает на вход двойной указатель на массив пикселей, координату начала горизонтальной линии, координату конца горизонтальной линии, координату, по которой проводится горизонтальная линия, структуру цвета пикселя, указатель на структуру информационного заголовка BMP файла.
10. *void yLine(Rgb** rgb, int x, int y1, int y2, Rgb color, BitmapInfoHeader* bmif)* – метод рисования вертикальной линии пикселей заданного цвета между внутренним и внешним радиусом по координате x. Принимает на вход двойной указатель на массив пикселей, координату начала вертикальной линии, координату конца вертикальной линии, координату, по которой проводится вертикальная линия, структуру цвета пикселя, указатель на структуру информационного заголовка BMP файла.
11. *void transformation(Rgb* pixel, char* color)* - метод задачи цвета пикселя. Принимает на вход указатель на структуру цвета пикселя, строку с описанием цвета.

12. *void draw_circle(Rgb** rgb, Rgb pixel, int x1, int y1, int xo, int xi, int y, int erro, int erri, BitmapInfoHeader* bmif)* – метод рисования окружности с использованием алгоритма Брезенхэма. Принимает на вход двойной указатель на массив пикселей, структуру цвета пикселя, координаты центра окружности, внешний радиус, внутренний радиус, текущую координату y на окружности, значения ошибок внутреннего радиуса для коррекции окружности, значения ошибок внешнего радиуса для коррекции окружности, указатель на структуру информационного заголовка BMP файла.
13. *void Circle(Rgb** rgb, BitmapInfoHeader *bmif, Options option, Errors* err)* – метод обработки флага –circle. Принимает на вход двойной указатель на массив пикселей, указатель на структуру информационного заголовка BMP файла, структуру опций, указатель на структуру ошибок.
14. *void swap(Rgb *pixel1, Rgb *pixel2)* – метод обмена значений цветов двух пикселей. Принимает на вход указатель на структуру цвета одного пикселя и указатель на структуру цвета другого пикселя.
15. *void Mirror(Rgb** rgb, BitmapInfoHeader *bmif, Options option)* – метод отражения области изображения. Принимает на вход двойной указатель на массив пикселей, указатель на структуру информационного заголовка BMP файла, структуру опций.
16. *void Copy(Rgb** rgb, BitmapInfoHeader *bmif, Options option)* – метод копирования области изображения. Принимает на вход двойной указатель на массив пикселей, указатель на структуру информационного заголовка BMP файла, структуру опций.
17. *Errors processing_image(char* namefile, Options option)* – функция обработки изображения. Принимает на вход имя файла и структуру опций. Возвращает структуру ошибок.
18. *void processing_arc(int argc, char *argv[])* – метод обработки параметров командной строки. Принимает на вход количество аргументов командной строки, массив строк, содержащий аргументы командной строки.

19. *int main(int argc, char *argv[])* – главная функция. Принимает на вход количество аргументов командной строки, массив строк, содержащий аргументы командной строки.

Созданная программа разделена на модули, что позволило разбить сложную задачу на более мелкие и самостоятельные подзадачи, каждая функция отвечает за конкретную задачу, что упрощает понимание, поддержку, и читаемость кода.

Разработанный программный код см. в приложении А.

ТЕСТИРОВАНИЕ

Рисунок 1 – изображение для тестирования

1. Тестирование функции *circle*:

   Аргументы для запуска: ./cw --circle --center 300.300 --radius 100 --thickness 10 --color 255.0.0 input.bmp

 .

Рисунок 2 – результат работы функции *circle*

2. Тестирование функции *circle*:

   Аргументы для запуска: ./cw --circle --center 200.200 --radius 50 --thickness 10 --color 255.0.0 --fill --fill_color 255.255.0 input.bmp

Рисунок 3 – результат работы функции *circle*

3. Тестирование функции *mirror*:

   Аргументы для запуска: ./cw --mirror --axis x --left_up 100.100 --right_down 300.300 input.bmp

Рисунок 4 – результат работы функции *mirror*

4. Тестирование функции *mirror*:

   Аргументы для запуска: ./cw --mirror --axis y --left_up 100.100 --right_down 300.300 input.bmp

Рисунок 5 – результат работы функции *mirror*

5. Тестирование функции *copy*:

   Аргументы для запуска: ./cw --copy --left_up 50.100 --right_down 150.300 --dest_left_up 200.100 input.bmp

Рисунок 6 – результат работы функции *copy*

6. Тестирование функции *copy*:

   Аргументы для запуска: ./cw --copy --left_up 100.100 --right_down 300.300 --dest_left_up 50.400 input.bmp

Рисунок 7 – результат работы функции *copy*

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была создана программа на языке программирования C, осуществляющая обработку BMP изображения. В зависимости от выбранных опций, программа выполняет одну из поддерживаемых функций. Запуск программы и выбор опций осуществляется через CLI (command line interface).

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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

#pragma pack (push, 1) // для корректного считывания данных выполнения
выравнивания на 1 байт

// структура для хранения BitmapFileHeader (загаловок файла)
typedef struct {
    unsigned short signature; // сигнатура BMP файла
    unsigned int filesize; // общий размер файла в байтах
    unsigned short reserved1; // зарезервированно для будущего
использования
    unsigned short reserved2; // зарезервированно для будущего
использования
    unsigned int pixelArrOffset; // смещение (позиция) начала массива
пикселей относительно начала файла
} BitmapFileHeader;

// структура для хранения BitmapInfoHeader (информационный загаловок)
typedef struct {
    unsigned int headerSize; // размер информационного заголовка
    unsigned int width; // ширина ихображения в пикселях
    unsigned int height; // высота изображения в пикселях
    unsigned short planes; // кол-во плоскостей изображения
    unsigned short bitsPerPixel; // кол-во бит на пиксель
    unsigned int compression; // тип сжатия
    unsigned int imageSize; // размер изображения в байтах
    unsigned int xPixelsPerMeter; // кол-во пикселей на метр по
горизонтали
    unsigned int yPixelsPerMeter; // кол-во пикселей на метр по вертикали
    unsigned int colorsInColorTable; // кол-во цветов в цветовой таблице
    unsigned int importantColorCount; // кол-во важных цветов для
изображения
} BitmapInfoHeader;

// структура для хранения цветовой компаненты пикселя
typedef struct {
    unsigned char b; // синяя компанента
    unsigned char g; // зеленная компанента
    unsigned char r; // красная компанента
} Rgb;

#pragma pack(pop) // восстановление предыдущего уровня выравнивания
структур

typedef struct {
```

```
    int circle;
    int mirror;
    int copy;
    int info;
    int help;

    char* input_name;
    char* output_name; //"out.bmp"

    int center [2]; //x.y
    int radius; // >0
    int thickness; // >0
    char* color; //rrrr.ggg.bbb
    int fill; // true/false
    char* fill_color; // rrr.ggg.bbb

    char*  axis; // x/y
    int left_up [2]; // x.y
    int right_down [2]; // x.y

    int dest_left_up [2]; // x.y

} Options;

// структура ошибок
typedef struct {
    int value;
    char* message_error;
} Errors;

// метод вывода справки
void reference(){
    printf("Reference.\n");
    printf("Usage:\n");
    printf("--circle\n");
    printf("--center x.y --radius num > 0 --thickness num > 0 --color
rrr.ggg.bbb --fill true/false --fill_color rrr.ggg.bbb\n");
    printf("--mirror\n");
    printf("--axis x/y --left_up x.y --right_down x.y\n");
    printf("--copy\n");
    printf("--left_up x.y --right_down x.y --dest_left_up x.y\n");
}

// методы для печати информации о заголовке файла
// каждый параметр распечатывается с указанием 16го и 10го представления
void print_file_header(BitmapFileHeader *header){
    // вывод сигнатуры файла
    printf("signature:\t%x (%hu)\n", header->signature, header-
>signature);
    // вывод размера файла
    printf("filesize:\t%x (%u)\n", header->filesize, header->filesize);
    // вывод зарезервированных значений
    printf("reserved1:\t%x (%hu)\n", header->reserved1, header-
>reserved1);
    printf("reserved2:\t%x (%hu)\n", header->reserved2, header-
>reserved2);
    // вывод смещения до массива пикселей
```

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

// метод для печати информации из заголовка изображения
void print_info_header(BitmapInfoHeader *header){
    // вывод размера заголовка
    printf("headerSize:\t%x (%u)\n", header->headerSize, header-
>headerSize);
    // вывод ширины изображения
    printf("width: \t%x (%u)\n", header->width, header->width);
    // вывод высоты изображения
    printf("height: \t%x (%u)\n", header->height, header->height);
    //вывод числа плоскостей
    printf("planes: \t%x (%hu)\n", header->planes, header->planes);
    // вывод числа бит на пиксель
    printf("bitsPerPixel:\t%x (%hu)\n", header->bitsPerPixel, header-
>bitsPerPixel);
    // вывод метода сжатия изображения
    printf("compression:\t%x (%u)\n", header->compression, header-
>compression);
    // вывод размера изображения в байтах
    printf("imageSize:\t%x (%u)\n", header->imageSize, header-
>imageSize);
    // вывод разрешения по горизонтали
    printf("xPixelsPerMeter:\t%x (%u)\n", header->xPixelsPerMeter,
header->xPixelsPerMeter);
    // вывод разрешения по вертикали
    printf("yPixelsPerMeter:\t%x (%u)\n", header->yPixelsPerMeter,
header->yPixelsPerMeter);
    // вывод кол-ва цветов в цветовой таблице
    printf("colorsInColorTable:\t%x (%u)\n", header->colorsInColorTable,
header->colorsInColorTable);
    // вывод кол-ва важных цветов
    printf("importantColorCount:\t%x (%u)\n", header-
>importantColorCount, header->importantColorCount);
}

// метод записи данных в файл
void write_bmp(char* file_name, Rgb **arr, BitmapFileHeader *bmfh,
BitmapInfoHeader *bmif, Errors* err){
    FILE *ff = fopen(file_name, "wb"); // открытие файла для записи в
бинарном рижиме

    // обработка ошибки открытия файла
    if (ff == 0) {
        err->value = 1;
        err->message_error = "Error: opening the file for writing.\n";
        fclose(ff); // закрытие файла
    }

    // запись заголовка файла и информации об изображении
    fwrite(bmfh, 1, sizeof(BitmapFileHeader), ff);
    fwrite(bmif, 1, sizeof(BitmapInfoHeader), ff);

    unsigned int H = bmif->height; // высота изображения
    unsigned int W = bmif->width; // ширина изображения
```

```
    fseek(ff, bmfh->pixelArrOffset, SEEK_SET); // перемещение указателя
файла на начало массива пикселей

    // запись массива пикселей покадрово (по строкам) в файл
    for(int i = 0; i < H; i++){
        fwrite(arr[i], 1, W * sizeof(Rgb) + ((4-(W*sizeof(Rgb))%4)%4),
ff);
    }

    fclose(ff); // закрытие файла после записи
}

// функция чтения файла
Rgb **read_bmp(char* file_name, BitmapFileHeader *bmfh, BitmapInfoHeader
*bmif, Errors *err) {
    FILE *f = fopen(file_name, "rb"); // открытие файла для чтения в
бинарном режиме

    // обработка ошибки открытия файла
    if (f == 0) {
        err->value = 1;
        err->message_error = "Error: opening the file for reading.\n";
    }

    if(err->value == 1) {
        Rgb **arr = NULL;
        return arr;
    } else {
        fread(bmfh, 1, sizeof(BitmapFileHeader), f); // считывания данных
структуры BitmapFileHeader и сохранение их в bmfh

        // проверка сигнатуры файла на соответствие формату BMP или его
версиям
        if (bmfh->signature != 0x4D42) {
            err->value = 1;
            err->message_error = "Error: file is not a BMP format.\n";
            fclose(f);
        }

        fread(bmif, 1, sizeof(BitmapInfoHeader), f); // считывание данных
структуры BitmapInfoHeader и сохранение их в bmif

        unsigned int H = bmif->height; // высота изображения
        unsigned int W = bmif->width; // ширина изображения
        fseek(f, bmfh->pixelArrOffset, SEEK_SET); // перемещение
указателя файла на начало массива пикселей

        Rgb **arr = malloc(H * sizeof(Rgb*)); // выделение памяти под
массив пикселей

        // обработка ошибки выделения памяти
        if (arr == NULL) {
            err->value = 1;
            err->message_error = "Error: failed to allocate memory.\n";
        }

        for (int i = 0; i < H; i++) {
```

```
            arr[i] = malloc(W * sizeof(Rgb) + ((4-(W*sizeof(Rgb))%4)%4));
// выделение памяти для строки пикселей с учетом выравнивания

            // обработка ошибки выделения памяти
            if (arr[i] == NULL) {
                err->value = 1;
                err->message_error = "Error: failed to allocate memory.\
n";
            }

            // считывание данных пикселей для текущей строки
            fread(arr[i], 1, W*sizeof(Rgb) + ((4-(W*sizeof(Rgb))%4)%4),
f);
        }

        fclose(f); // закрытие файла
        return arr;
    }
}

// очистка памяти, выделенной под массив пикселей
void freeRgbArr(Rgb** arr, unsigned int H){
    for(int i=0; i < H; i++){
        free(arr[i]);
    }
    free(arr);
}

// метод преобразования координат
void coordinate_conversion(int* mas, char* coordinate){
    char* token = strtok(coordinate, ".");
    mas[0] = atoi(token);
    token = strtok(NULL, ".");
    mas[1] = atoi(token);
}

// метод проверки параметров флагов на ошибки
void check_parametres(Options* option, BitmapInfoHeader* bmif, Errors
*err){
    // если окружность
    if(option->circle == 1){
        option->center[1] = bmif->height - option->center[1]; // переход
к системе координат с центром в левом нижнем углу
        // проверка координат центра
        if((option->center[0] < 0) || (option->center[1] < 0) || (option-
>center[0] > bmif->width) || (option->center[1] > bmif->height)){
            err->value = 1;
            err->message_error = "Error: coordinates aren't correct.\n";
        }
        // проверка радиуса
        if( option->radius <= 0 ){
            err->value = 1;
            err->message_error = "Error: radius isn't correct.\n";
        }
        // проверка толщины линии
        if(option->thickness <= 0){
            err->value = 1;
            err->message_error = "Error: thickness isn't correct.\n";
```

```
        }
    }
    // если отражение
    if(option->mirror == 1){
        // переход к системе координат с центром в левом нижнем углу
        option->left_up[1] = bmif->height - option->left_up[1];
        option->right_down[1] = bmif->height - option->right_down[1];

        // проверка оси отражения
        if((strcmp(option->axis, "x") != 0) && (strcmp(option->axis, "y")
!= 0) ){
            err->value = 1;
            err->message_error = "Error: axis isn't correct.\n";
        }
        // провека координат  левого верхнего угла области
        if((option->left_up[0] < 0) || (option->left_up[1] < 0) ||
(option->left_up[0] > bmif->width) || (option->left_up[1] > bmif-
>height)){
            err->value = 1;
            err->message_error = "Error: coordinates aren't correct.\n";
        }
        // проверка координат правого нижнего угла области
        if((option->right_down[0] < 0) || (option->right_down[1] < 0) ||
(option->right_down[0] > bmif->width) || (option->right_down[1] > bmif-
>height)){
            err->value = 1;
            err->message_error = "Error: coordinates aren't correct.\n";
        }
    }

    // если копирование
    if(option->copy == 1){
        // переход к системе координат с центром в левом нижнем углу
        option->left_up[1] = bmif->height - option->left_up[1];
        option->right_down[1] = bmif->height - option->right_down[1];
        option->dest_left_up[1] = bmif->height - option->dest_left_up[1];
        // провека координат  левого верхнего угла области
        if((option->left_up[0] < 0) || (option->left_up[1] < 0) ||
(option->left_up[0] > bmif->width) || (option->left_up[1] > bmif-
>height)){
            err->value = 1;
            err->message_error = "Error: coordinates aren't correct.\n";
        }
        // проверка координат правого нижнего угла области
        if((option->right_down[0] < 0) || (option->right_down[1] < 0) ||
(option->right_down[0] > bmif->width) || (option->right_down[1] > bmif-
>height)){
            err->value = 1;
            err->message_error = "Error: coordinates aren't correct.\n";
        }
        // проверка координат левого верхнего угла области назначения
        if((option->dest_left_up[0] < 0) || (option->dest_left_up[1] < 0)
|| (option->dest_left_up[0] > bmif->width) || (option->dest_left_up[1] >
bmif->height)){
            err->value = 1;
            err->message_error = "Error: coordinates aren't correct.\n";
        }
    }
```

```
}

// метод рисования горизонтальной линии пикселей заданного цвета между
внутренним и внешним радиусом по координате y
void xLine(Rgb** rgb, int x1, int x2, int y, Rgb color, BitmapInfoHeader*
bmif) {
    while (x1 <= x2) {
        if (y >= 0 && y < bmif->width  && x1 < bmif->height) {
            rgb[x1][y] = color;
        }
        x1++;
    }
}

// метод рисования вертикальной линии пикселей заданного цвета между
внутренним и внешним радиусом по координате x
void yLine(Rgb** rgb, int x, int y1, int y2, Rgb color, BitmapInfoHeader*
bmif) {
    while (y1 <= y2 ) {
        if (x >= 0 && x < bmif->height && y1 < bmif->width) {
            rgb[x][y1] = color;
        }
        y1++;
    }
}

// метод задачи цвета пикселя
void transformation(Rgb* pixel, char* color){
    char* token = strtok(color, ".");
    int red = atoi(token);
    token = strtok(NULL, ".");
    int green = atoi(token);
    token = strtok(NULL, ".");
    int blue = atoi(token);
    pixel->b = blue;
    pixel->g = green;
    pixel->r = red;
}

// метод рисования окружности
void draw_circle(Rgb** rgb, Rgb pixel, int x1, int y1, int xo, int xi,
int y, int erro, int erri, BitmapInfoHeader* bmif){
    while ((xo >= y)) {
        xLine(rgb, x1 + xi, x1 + xo, y1 + y, pixel, bmif);
        yLine(rgb, x1 + y, y1 + xi, y1 + xo, pixel, bmif);
        xLine(rgb, x1 - xo, x1 - xi, y1 + y, pixel, bmif);
        yLine(rgb, x1 - y, y1 + xi, y1 + xo, pixel, bmif);
        xLine(rgb, x1 - xo, x1 - xi, y1 - y, pixel, bmif);
        yLine(rgb, x1 - y, y1 - xo, y1 - xi, pixel, bmif);
        xLine(rgb, x1 + xi, x1 + xo, y1 - y, pixel, bmif);
        yLine(rgb, x1 + y, y1 - xo, y1 - xi, pixel, bmif);

        y++;

        if (erro < 0) {
            erro += 2 * y + 1;
        } else {
            xo--;
```

```
            erro += 2 * (y - xo + 1);
        }

        if (y > xi) {
            xi = y;
        } else {
            if (erri < 0) {
                erri += 2 * y + 1;
            } else {
                xi--;
                erri += 2 * (y - xi + 1);
            }
        }
    }
}

// метод обработки флага --circle
void Circle(Rgb** rgb, BitmapInfoHeader *bmif, Options option, Errors*
err) {
        // получение информации о цвете окружности
        Rgb pixel; // цвет линии
        transformation(&pixel, option.color);

        // проверка на соответствие цвета линии
        if((pixel.b < 0 || pixel.b > 255) || (pixel.g < 0 || pixel.g >
255) ||  (pixel.r < 0 || pixel.r > 255)){
            err->value = 1;
            err->message_error = "Error: color isn't correct.\n";
        }

        if(err->value != 1){
            // проверка на наличие заливки
            if(option.fill == 1){
                // получение информации о цвете заливки
                Rgb pixel_fill; // цвет заливки
                transformation(&pixel_fill,option.fill_color);

                // проверка на соответствие цвета заливки
                if((pixel_fill.b < 0 || pixel_fill.b > 255) ||
(pixel_fill.g < 0 || pixel_fill.g > 255) ||  (pixel_fill.r < 0 ||
pixel_fill.r > 255)){
                    err->value = 1;
                    err->message_error = "Error: color isn't correct.\n";
                }

                if(err->value != 1){
                    // отрисовка залитой окружности
                    draw_circle(rgb, pixel_fill, option.center[1],
option.center[0], option.radius, 0, 0, 1 - option.radius, 1, bmif);
                }
            }
            // отрисовка окружности без заливки
            draw_circle(rgb, pixel, option.center[1], option.center[0],
option.radius + option.thickness / 2, option.radius - option.thickness /
2, 0, 1 - (option.radius + option.thickness / 2), 1 - (option.radius -
option.thickness / 2), bmif);
        }
}
```

```
// метод обмена значений
void swap(Rgb *pixel1, Rgb *pixel2) {
    Rgb temp = *pixel1;
    *pixel1 = *pixel2;
    *pixel2 = temp;
}

// метод отражения области
void Mirror(Rgb** rgb, BitmapInfoHeader *bmif, Options option){
// знаем ось, координаты левого верхнего угла, координаты правого нижнего
угла
/*  x0 , y0                     x , y0

    x0 , y                      x , y
*/
if(strcmp(option.axis, "y") == 0){
    int x0 = option.left_up[0];
    int x = option.right_down[0];
    while(x0 <= x){
        int y0 = option.left_up[1] - 1;
        int y = option.right_down[1] - 1;
        while(y <= y0){
            swap(&rgb[y][x0], &rgb[y0][x0]);
            y++;
            y0--;
        }
        x0++;
    }
}
if(strcmp(option.axis, "x") == 0){
    int y0 = option.left_up[1];
    int y = option.right_down[1];
    while(y0 >= y){
        int x0 = option.left_up[0];
        int x = option.right_down[0];
        while(x0 <= x){
            swap(&rgb[y0][x0], &rgb[y0][x]);
            x0++;
            x--;
        }
        y0--;
    }
}
}

// метод копирование области
void Copy(Rgb** rgb, BitmapInfoHeader *bmif, Options option){
// знаем ось, координаты левого верхнего угла источника, координаты
правого нижнего угла источника, координаты правого верхнего угла области
назначения
/*  x0 , y0                     x , y0              xn , yn
* , *
                                                ->
    x0 , y                      x , y               * , *
* , *
*/
// получаем значения координат
```

```
int x0 = option.left_up[0];
int y0 = option.left_up[1];
int x = option.right_down[0];
int y = option.right_down[1];
int xn = option.dest_left_up[0];
int yn = option.dest_left_up[1];
int h = y0 - y + 1; // высота области копирования
int w = x - x0 + 1; // ширина области копирования

Rgb **buffer = malloc(h * sizeof(Rgb*)); // выделение памяти для хранения
буфера копирования
for(int i = 0; i < h; i++){
    buffer[i] = malloc(w * sizeof(Rgb) + ((4-(w*sizeof(Rgb))%4)%4)); //
выделение памяти для строки пикселей  в буффере с учетом выравнивания
}

// копирование заданной области во временный буфер
for(int i = 0; i < h; i++){
    for(int j = 0; j < w; j++){
        buffer[i][j] = rgb[y0-i][x0+j]; // копирование во временный буфер
    }
}

int lim_x, lim_y; // ограничение координат копирования по ширине и высоте
изображения
if(xn + w > bmif->width){
    lim_x = bmif->width;
} else {
    lim_x = xn + w;
}
if(yn - h < 0){
    lim_y = 0;
} else {
    lim_y = yn - h;
}

// вставка скопированной области
for(int i = yn; i > lim_y; i--){
    for(int  j = xn; j < lim_x; j++){
        rgb[i][j] = buffer[yn-i][j-xn];
    }
}
// освобождение памяти, выделенной для хранение временного буфера
freeRgbArr(buffer, h);

}

// метод обработки изображения
Errors processing_image(char* namefile, Options option){
    // инициализация переменных для хранения заголовка изображения
    BitmapFileHeader bmfh;
    BitmapInfoHeader bmif;

    Errors err;
    err.value = 0;
    err.message_error = "NULL";

    // чтение данных об изображении в массив пикселей
```

```
    Rgb** rgb = read_bmp(namefile, &bmfh, &bmif, &err);

    if(err.value != 1){
        // обработка заданной опции
        if(option.info == 1){ // печать информации об изображениии
            print_file_header(&bmfh);
            print_info_header(&bmif);
        }

        if(option.circle == 1){ // рисование окружности
            //проверка параметров флагов
            check_parametres(&option, &bmif, &err);
            if(err.value != 1){
                Circle(rgb, &bmif, option, &err);
            }
        }

        if(option.mirror == 1){ // отражение области
            //проверка параметров флагов
            check_parametres(&option, &bmif, &err);
            if(err.value != 1){
                Mirror(rgb, &bmif, option);
            }
        }

        if(option.copy == 1){ // копирование области
            //проверка параметров флагов
            check_parametres(&option, &bmif, &err);
            if(err.value != 1){
                Copy(rgb, &bmif, option);
            }
        }


        // запись измененного изображения в файл
        write_bmp(option.output_name, rgb, &bmfh, &bmif, &err);

        // освобождение памяти, выделенной под хранение массива пикселей
        freeRgbArr(rgb, bmif.height);
    }

    return err;
}

// метод обработки параметров командной сторки
void processing_arc(int argc, char *argv[]){
    // определение длинных опций
    struct option long_opts[] = {
        {"help", no_argument, 0,'h'},
        {"info",no_argument,0, 6},
        {"input", required_argument, 0, 'i'},
        {"output", required_argument, 0, 'o'},

        {"circle", no_argument, 0, 'C'},
        {"center", required_argument, 0, 'O'},
        {"radius", required_argument, 0, 'r'},
        {"thickness", required_argument, 0, 't'},
        {"color", required_argument, 0, 'H'}, // h -> hue оттенок
```

```
        {"fill", no_argument, 0, 'f'},
        {"fill_color", required_argument, 0, 'p'}, // p -> pouring
разливка

        {"mirror", no_argument, 0, 'm'},
        {"axis", required_argument, 0, 'a'},
        {"left_up", required_argument, 0, 'u'},
        {"right_down", required_argument, 0, 'd'},

        {"copy", no_argument, 0, 'c'},
        {"dest_left_up", required_argument, 0, 'n'}, // n -> nook угол
        {0,0,0,0}, // маркер конца списка длинных опция для getopt_long()
    };

    Options option =  {
        .circle = 0,
        .mirror = 0,
        .copy = 0,
        .info = 0,
        .help = 0,
        .input_name = argv[argc-1],
        .output_name = "out.bmp",
        .center[0] = 0,
        .center[1] = 0,
        .radius = 0,
        .thickness = 0,
        .color = NULL,
        .fill = 0,
        .fill_color = NULL,
        .axis = NULL,
        .left_up[0] = 0,
        .left_up[1] = 0,
        .right_down[0] = 0,
        .right_down[1] = 0,
        .dest_left_up[0] = 0,
        .dest_left_up[1] = 0,
    };

    int opt;
    opterr = 0; // отключение переменной, контролирующей ошибки в
параметрах командной строки
    Errors error;
    error.value = 0;
    error.message_error = "";

    while((opt = getopt_long(argc, argv, "hi:o:CO:r:t:H:fp:ma:u:d:cn:",
long_opts, NULL)) != -1){
        switch (opt)
        {
        case 'h': // вывод справки
            option.help = 1;
            break;
        case 6: // вывод информации об изображении
            option.info = 1;
            break;
        case 'i': // задаёт имя входного изображения
            option.input_name = optarg;
            break;
```

```
        case 'o': // задаёт имя выходного изображения
            option.output_name = optarg;
            break;
        case 'C': // вызов рисования окружности
            option.circle = 1;
            break;
        case 'O': // задаёт центр окружности
            coordinate_conversion(option.center, optarg);
            break;
        case 'r': // задаёт радиус окружности
            option.radius = atoi(optarg);
            break;
        case 't': // задаёт толщину линии
            option.thickness = atoi(optarg);
            break;
        case 'H': // задаёт цвет линии
            option.color = optarg;
            break;
        case 'f': // наличии заливки
            option.fill = 1;
            break;
        case 'p': // задаёт цвет заливки
            option.fill_color = optarg;
            break;
        case 'm': // вызов отражения заданной области
            option.mirror = 1;
            break;
        case 'a': // задаёт ось отображения
            option.axis = optarg;
            break;
        case 'u': // задаёт координаты левого верхнего угла области
            coordinate_conversion(option.left_up, optarg);
            break;
        case 'd': // задаёт координаты правого нижнего угла области
            coordinate_conversion(option.right_down, optarg);
            break;
        case 'c': // вызов копирования заданной области
            option.copy = 1;
            break;
        case 'n': // задаёт координаты левого верхнего угла области-
назначения
            coordinate_conversion(option.dest_left_up, optarg);
            break;
        default:
            error.value = 1;
            error.message_error = "Error: couldn't recognize given flag.\
n";
            break;
        }
    }
        // Проверка на совпадение имен входного и выходного файлов
        if (strstr(option.input_name, option.output_name)) {
            error.value = 1;
            error.message_error = "Can't use the same input and output
file names.\n";
        }
        // проверка на отсутсвие флагов
        if(optind == 1){
```

```
            error.value = 1;
            error.message_error = "Error: flags aren't set.\n";
        }

        if(error.value == 1){
           printf("%s",error.message_error);

        } else {
            if(option.help != 1){
                // обработка изображения
                error = processing_image(option.input_name, option);
                if (error.value) printf("%s", error.message_error);
            }
            else {
                reference();
            }
        }

}

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

    // информация о курсовой работе
    printf("Course work for option 4.1, created by Darya Sinitskaya.\n");

    // вызов обработки параметров командной строки
    processing_arc(argc, argv);

    return 0;
}
```