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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Работа с изображениями

Студент гр. 3342 _____ Лапшов К. Н.

Преподаватель _____ Глазунов С. А.

Санкт-Петербург

2024

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент: Лапшов К. Н.

Группа: 3342

Тема работы: Работа с изображениями

Исходные данные:
Вариант 5.5
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 следующие функции по обработке изображений:

- Инвертировать цвета в заданной окружности. Флаг для выполнения данной операции: `--inverse_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 0

- Обрезка изображения. Флаг для выполнения данной операции: `--trim`. Требуется обрезать изображение по заданной области. Область определяется:
  1. Координатами левого верхнего угла. Флаг `--left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y
  2. Координатами правого нижнего угла. Флаг `--right_down`, значение задаётся в формате `right.down`, где right – координата по x, down – координата по y
- Рисование треугольника. Флаг для выполнения данной операции: `--triangle`. Треугольник определяется:
  1. Координатами его вершин. Флаг `--points`, значение задаётся в формате `x1.y1.x2.y2.x3.y3` (точки будут (x1; y1), (x2; y2) и (x3; y3)), где x1/x2/x3 – координаты по x, y1/y2/y3 – координаты по y
  2. Толщиной линий. Флаг `--thickness`. На вход принимает число больше 0
  3. 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)
  4. Треугольник может быть залит или нет. Флаг `--fill`. Работает как бинарное значение: флага нет – false , флаг есть – true.
  5. Цветом которым он залит, если пользователем выбран залитый. Флаг `--fill_color` (работает аналогично флагу `--color`)
- Рисование отрезка. Флаг для выполнения данной операции: `--line`. Отрезок определяется::
  1. координатами начала. Флаг `--start`, значение задаётся в формате

`x.y`, где x – координата по x, y – координата по y

2. координатами конца. Флаг `--end` (аналогично флагу `--start`)
3. 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)
4. 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

Разделы пояснительный записки: «Содержание», «Введение», «Структуры», «Функции», «Заключение», «Список использованных источников», «Приложение А. Примеры работы программы», «Приложение Б. Исходный код программы».

Предполагаемый объем пояснительной записки:

Не менее 25 страниц.

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 15.05.2024

Дата защиты реферата: 22.05.2024

Студент _____ Лапшов К. Н.

Преподаватель _____ Глазунов С. А.

АННОТАЦИЯ

Цель данной курсовой работы заключается в разработке программы на языке C, предназначенной для обработки изображений в формате BMP.

Курсовая представляет набор компонентов для обработки файлов формата BMP, включающих функции обрезки, рисования линии и треугольника, а также функцию инвертирования цвета в заданной окружности. Структура курсовой включает в себя файлы исходного кода, заголовочные файлы, а также Makefile для сборки проекта. После завершения сборки проекта можно проводить различные манипуляции с файлами BMP, задавая соответствующие параметры через командную строку. Затем программа осуществляет необходимую логику для выполнения указанного функционала и сохраняет результат в новом файле.

Пример работы программы приведен в приложении А.

Исходный код программы приведен в приложении Б.

SUMMARY

The purpose of this course work is to develop a C program designed for image processing in BMP format.

The coursework is a set of components for processing BMP files, including cropping functions, drawing lines and triangles, as well as the function of inverting colors in a given circle. The course structure includes source code files, header files, and a Makefile for building the project. After completing the project build, you can perform various manipulations with BMP files by setting the appropriate parameters via the command line. Then the program implements the necessary logic to perform the specified functionality and saves the result in a new file.

An example of how the program works is given in Appendix A.

The source code of the program is given in Appendix B.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы: написать программу на языке C для обработки изображения. Для этого требется реализовать:

- Функции загрузки, хранения и записи изображений их/в файл;
- Возможность ввода аргументов через командную строку;
- Реализацию функций для рисования на изображении.

1. СТРУКТУРЫ

Для взаимодействия с изображением формата BPM и для облегчения написания кода были реализованы следующие структуры: BitmapFileHeader, BitmapInfoHeader, RGB, BMP, Point, Option.

1.1 Структура BitmapFileHeader

Структура BitmapFileHeader содержит информацию о файле в целом, такую как тип файла, размер файла, местоположение данных пикселей и т.д. Она содержит следующие поля:

- ` bfSign`: Отметка для отличия формата от других (сигнатура формата).

Должна быть равна 'BM' для файлов BMP.

- ` bfSize`: Размер файла в байтах.

- `bfReserved1`: Зарезервированное поле, которое должно быть равно нулю.

- `bfReserved2`: Зарезервированное поле, которое должно быть равно нулю.

- `bfArrOffset `: Смещение от начала файла до начала данных пикселей.

Это поле указывает, где в файле начинаются данные массива пикселей.

Эта структура необходима, потому что она предоставляет базовую информацию, необходимую для чтения и интерпретации файла BMP.

1.2 Структура BitmapInfoHeader

Структура BitmapInfoHeader следует непосредственно после BitmapFileHeader и содержит информацию о самом изображении, такую как его размеры, формат цвета, разрешение и т.д. Включает в себя следующие поля:

- `biSize`: Размер структуры BitmapInfoHeader в байтах.

- `biWidth`: Ширина изображения в пикселях.

- `biHeight`: Высота изображения в пикселях.

- `biPlanes`: Количество цветовых плоскостей.

- `biBitSize`: Количество бит на пиксель (битовая глубина).

- `biImageSize`: Размер изображения в байтах.

- `biXPixelsPerMeter`: Горизонтальное разрешение изображения в пикселях на метр.

- `biYPixelsPerMeter`: Вертикальное разрешение изображения в пикселях на метр.

- `biClrTotal`: Количество цветов в цветовой палитре.

- `biClrImportant`: Количество используемых важных цветов.

1.3 Структура RGB

Структура RGB определяет формат пикселя в изображении, в которой хранятся значения красной, зеленой и синей компоненты цвета пикселя. Включает в себя следующие поля:

- `b`: Синяя компонента пикселя.

- `g`: Зеленая компонента пикселя.

- `r`: Красная компонента пикселя.

1.4 Структура BMP

Структура BMP определяет формат файла BMP. Реализована для упрощения взаимодействия с файлом изображения. Включает в себя следующие поля:

- ` bmfh`: Структура типа BitmapFileHeader.

- ` bmih`: Структура типа BitmapInfoHeader.

- `img`: Массив пикселей изображения.

1.5 Структура Point

Структура BMP определяет точку в двумерном пространстве с целочисленными координатами x и y. Реализована для упрощения взаимодействия с координатами на изображении. Включает в себя следующие поля:

- ` x`: Координаты точки по ширине относительно массива пикселей.

- ` y`: Координаты точки по высоте относительно массива пикселей.

1.6 Структура Option

Эта структура содержит множество полей, которые соответствуют флагам, указанным пользователем при запуске программы. Из нее можно извлечь сведения о том, был ли указан конкретный флаг и какое значение он имеет.

2. ФУНКЦИИ

2.1 Функция main

Функция `main` является основной функцией программы, которая управляет аргументами командной строки и выполняет операции над изображением. Она принимает параметры `argc` (количество аргументов командной строки) и `argv` (массив строк, содержащий эти аргументы).

Внутри функции происходят следующие действия:

Инициализируется структура `Option`, в которой хранятся все поступающие аргументы из командной строки, с помощью функции `getCliFlagsInfo`, в которой происходит обработка аргументов.

После этого читается вся информация о входном изображении с помощью функции `readBMP` и сохраняется в структуру BMP.

Далее выполняется выполнение поставленной задачи на основе предоставленных параметров с помощью функции `choiceTask`.

В конце, измененное изображение записывается в выходной BMP-файл с помощью функции `saveBMP`.

2.2 Функции для работы с CLI интерфейсом

Функция `getCliFlagsInfo` обрабатывает флаги и их параметры из командной строки, переданные программе. Создает, и с помощью библиотеки `getopt.h`, заполняет структуру `Option`. В функцию передается количество аргументов командной строки `argc`, массив строк, содержащий аргументы командной строки `argv`. Функция возвращает структуру типа `Option`.

Функция `getPointValue` преобразует координаты, предоставленные в виде строки, в структуру типа `Point`, содержащую координаты x и y. Возвращает заполненную структуру.

Функция `getRGBValue` преобразует цвет, предоставленный в виде строки, в структуру типа `RGB`, содержащую красную, зеленую и синюю компоненты цвета. Возвращает заполненную структуру.

Функция `getPointsValue` преобразует массив координат, предоставленный в виде строки, в массив структур типа ` Point`, содержащий координаты X и Y. Возвращает указатель на этот массив.

Функция `getNonNegativeNumber` преобразует целочисленное значение, предоставленное в виде строки, в переменную типа `int`, проверяя, является ли число неотрицательным. Возвращает данную переменную.

Функция `printHelp` выводит справку о использовании программы и ее опций. Предоставляет информацию о курсовой работе, ее создателе и варианте. Описывает доступные опции включая длинные и короткие формы некоторых флагов, а так же небольшое объяснение о работе того или иного флага.

2.3 Функции для взаимодействия с BMP изображениями

Функция `getOffset` вычисляет смещение в байтах, которое нужно добавить к размеру одной строки изображения, для выравнивания его по границе 4 байт. Необходима для корректного считывания и записи изображения. Возвращает целочисленное значение.

Функция `readBMP` считывает BMP-изображение и сохраняет информацию о нем в структуру `BMP`. Принимает строку `file_name`, представляющую имя файла/путь к изображению. Проверяет изображение на соответствие формату BMP, а также на необходимую глубину цвета, в случае неверного формата данных, завершает работу приложения с ошибкой. Возвращает структуру `BMP`.

Функция `saveBMP` записывает BMP-изображение в файл. Принимает строку `file_name`, представляющую имя файла/путь к файлу, куда будет сохранено изображение, и указатель на структуру BMP, хранящую информацию о изображении для записи в файл. Ничего не возвращает.

Функция `createArrayOfPixels` выделяет память на новый массив пикселей изображения. Принимает на вход высоту и ширину изображения. Возвращает двумерный массив пикселей.

Функция `freeArrayOfPixels` освобождает память из под массива пикселей изображения. Принимает на вход ссылку на структуру изображения. Ничего не возвращает.

Функция `printInfo` выводит в консоль подробную информацию о BMP-изображении. На вход принимает структуру `BMP`, в которой и хранится вся подробная информация об изображении. Ничего не возвращает.

2.4 Функции для взаимодействия с пикселями изображения

Функция `setPixel` устанавливает пиксель с указанными значениями цвета. Принимает на вход указатель на структуру изображения, структуру цвета, а также структуру Point для указания конкретного пикселя на изображении.

Функция `getInvertPixelColor` инвертирует цвет заданного пикселя. Принимает на вход указатель на структуру изображения, а также структуру `Piont` для указания конкретного пикселя на изображении. Возвращает структуру инвертированного цвета пикселя.

Функция `pointInImage` проверяет, находится ли указанная точка в пределах изображения. Принимает на вход структуру `BitmapInfoHeader` и координаты точки, которую надо проверить. Ничего не возвращает.

Функция `checkCoordsInTriangle` проверяет, находится ли точка в заданном треугольнике. Необходима для работы функции ` triangle `. Принимает на вход точку, и три вершины треугольника.

Функция `swapInt` меняет значение целочисленных переменных. Необходима для работы функции `trim`. Принимает на вход указатели на две целочисленные переменные и меняет их значения местами. Ничего не возвращает.

2.5 Функции заданий

Функция ` fillCircle ` рисует заполненный круг на изображении. Принимает на вход указатель на структуру изображения, координату центра круга, значение радиуса, цвет заливки круга, а также булево значение для

указания необходимости инвертировании пикселей в круге. С помощью этой функции также реализуется рисование толщины в функции ` line`. Ничего не возвращает.

Функция `trim` обрезает изображение до указанной области. Принимает на вход указатель на структуру изображения, значение точки левого верхнего угла области, а также нижнего правого. Обрезает необходимую область и меняет информацию о изображении в заголовочных структурах типа ` BitmapFileHeader ` и ` BitmapInfoHeader `. Ничего не возвращает.

Функция `line` рисует линию на изображении. Принимает на вход указатель на структуру изображения, координаты начала и конца линии, целочисленное значение толщины линии, а также цвет линии. Логика рисования линии основана на алгоритме Брезенхема. В случае толщины больше одного пикселя, вместо `setPixel` используется `fillCircle`. Ничего не возвращает.

Функция `triangle` рисует закрашенный треугольник на изображении. Принимает на вход указатель на структуру изображения, массив трёх вершин треугольника, толщину ребра треугольника, булево значение необходимости заливки, а также цвет самой заливки. Логика рисования треугольника основана на функции `line`, заливка происходит с помощью функции ` checkCoordsInTriangle `. Ничего не возвращает.

Функция `choiceTask` реализует логику переключения задач на основе предоставленных аргументов. Принимает на вход указатель на структуру изображения, а также структуру `Option`, в которой хранятся все флаги и их значения. Ничего не возвращает.

2.6 Функция - обработчик ошибок

Функция `showError` реализует логику завершения программы с отображаемой ошибкой. Принимает на вход текстовое поле, которое необходимо показать перед аварийным завершением программы, а также код произошедшей ошибки. Ничего не возвращает.

Разработанный программный код см. в приложении Б.

ЗАКЛЮЧЕНИЕ

Была создана программа на языке C, которая считывает изображение и обрабатывает его в соответствии с запросами пользователя. В рамках этого были реализованы следующие возможности:

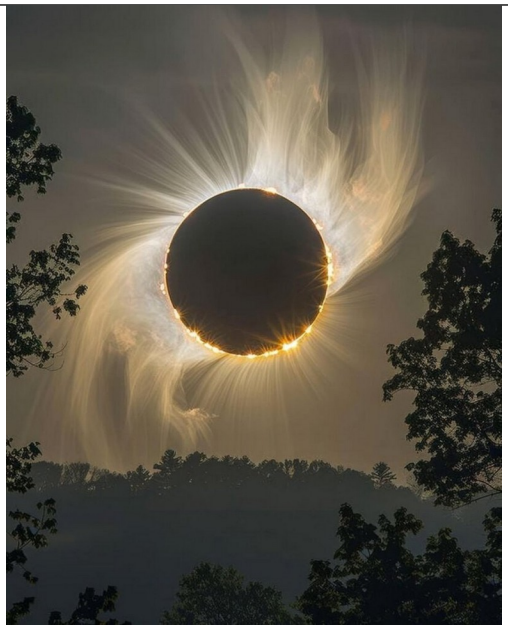
- Загрузка изображений из файла, их хранение и запись обратно в файл;
- Чтение аргументов из командной строки;
- Функции для редактирования загруженного изображения.

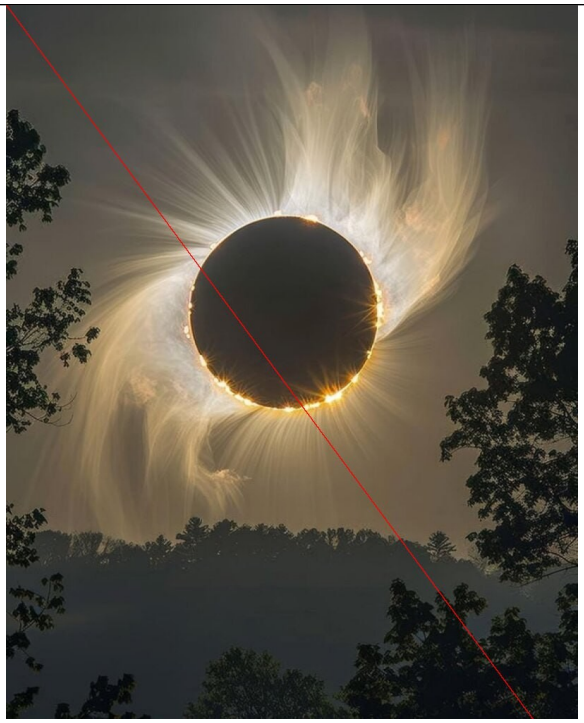
Для обработки аргументов командной строки использовалась библиотека getopt. Программа гарантированно завершает работу корректно при любых действиях пользователя, не выдавая ошибок.

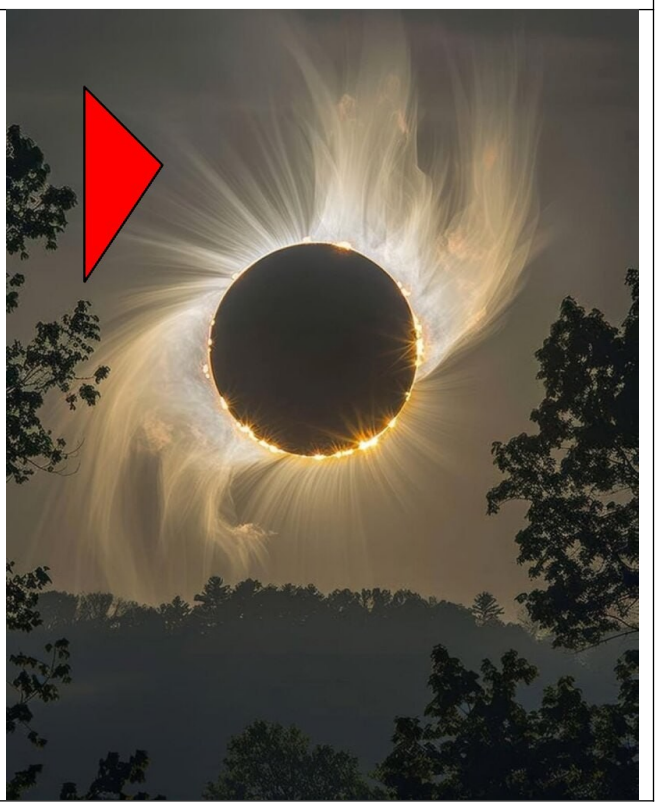
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Язык программирования С / Керниган Брайан, Ритчи Деннис. СПб.: "Финансы и статистика", 2003.
2. Викиучебник. URL: http://www.https://ru.wikibooks.org
3. БАЗОВЫЕ СВЕДЕНИЯ К ВЫПОЛНЕНИЮ КУРСОВОЙ РАБОТЫ ПО ДИСЦИПЛИНЕ «ПРОГРАММИРОВАНИЕ». ВТОРОЙ СЕМЕСТР. URL: https://se.moevm.info/lib/exe/fetch.php/courses:programming:programming_cw_metoda_2nd_course_last_ver.pdf.pdf

ПРИЛОЖЕНИЕ А
ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

Изображение test1.bmp	Изображение example.bmp
	

Входные данные	Выходные данные
./cw --line --start=0.0 --end=810.1100 --thickness=2 --color=255.0.0 example.bmp	

./cw --trim --left_up=100.100 --right_down=200.2000 test1.bmp	
./cw --inverse_circle --center=100.100 --radius=-30 test.bmp	Input data error!
./cw --triangle --points=100.100.200.200.100.350 --thickness=5 --color=0.0.0 --fill --fill_color=255.0.0 example.bmp	

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.c:

```
#include "structures.h"
#include "bmpInteraction.h"
#include "functionsInteraction.h"
#include "cliInterface.h"

int main(int argc, char *argv[]) {
    Option currentOption = getCliFlagsInfo(argc, argv);

    BMP currentBmp = readBMP(currentOption.inputFileName);
    choiceTask(&currentBmp, currentOption);
    saveBMP(currentOption.outputFileName, currentBmp);
    freeArrayOfPixels(&currentBmp);

    return 0;
}
```

Файл cliInerface.c:

```
#include "cliInterface.h"

Option
getCliFlagsInfo(int argc, char *argv[]) {
    Option options = {};
    options.outputFileName = "out.bmp";

    const char* shortOptions = "hi:o:";
    const struct option longOptions[] = {
        {"help", no_argument, NULL, 'h'},
        {"input", required_argument, NULL, 'i'},
        {"output", required_argument, NULL, 'o'},
        {"inverse_circle", no_argument, NULL, 400},
        {"trim", no_argument, NULL, 401},
        {"triangle", no_argument, NULL, 402},
        {"line", no_argument, NULL, 403},
        {"center", required_argument, NULL, 404},
```

```
            {"radius", required_argument, NULL, 405},
            {"left_up", required_argument, NULL, 406},
            {"right_down", required_argument, NULL, 407},
            {"points", required_argument, NULL, 408},
            {"thickness", required_argument, NULL, 409},
            {"color", required_argument, NULL, 410},
            {"fill", no_argument, NULL, 411},
            {"fill_color", required_argument, NULL, 412},
            {"start", required_argument, NULL, 413},
            {"end", required_argument, NULL, 414},
            {"info", no_argument, NULL, 415},
            {NULL, 0, NULL, 0}
        };


        if (argc == 1) {
            options.hasHelpOpt = 1;
            printHelp();
            exit(EXIT_SUCCESS);
        }

        int res;
             while ((res = getopt_long(argc, argv, shortOptions,
longOptions, NULL)) != -1) {
            switch (res) {
                //Флаги проверки информации о файле
                case 'h':
                    options.hasHelpOpt = true;
                    break;
                case 'i':
                    options.inputFileName = optarg;
                    options.hasInputOpt = true;
                    break;
                case 'o':
                    options.outputFileName = optarg;
                    options.hasOutputOpt = true;
                    break;

                //Флаги проверки функции
```

```
                case 400:
                     if(options.hasTrimOpt || options.hasTriangleOpt ||
options.hasLineOpt || options.hasInfoOpt) {
                         showError("Only one function can be choosed!",
ARGUMENTS_FAILURE);
                    }
                    options.hasInverseCircleOpt = true;
                    break;
                case 401:
                                    if(options.hasInverseCircleOpt  ||
options.hasTriangleOpt || options.hasLineOpt || options.hasInfoOpt) {
                         showError("Only one function can be choosed!",
ARGUMENTS_FAILURE);
                    }
                    options.hasTrimOpt = true;
                    break;
                case 402:
                                            if(options.hasTrimOpt   ||
options.hasInverseCircleOpt || options.hasLineOpt || options.hasInfoOpt)
{
                         showError("Only one function can be choosed!",
ARGUMENTS_FAILURE);
                    }
                    options.hasTriangleOpt = true;
                    break;
                case 403:
                     if(options.hasTrimOpt || options.hasTriangleOpt ||
options.hasInverseCircleOpt || options.hasInfoOpt) {
                         showError("Only one function can be choosed!",
ARGUMENTS_FAILURE);
                    }
                    options.hasLineOpt = true;
                    break;

                //Флаги проверки аргументов функции
                case 404:
                    if(!options.hasInverseCircleOpt) {
                          showError("No --inverse_circle flag for using
--center flag", ARGUMENTS_FAILURE);
```

```
                    }
                    options.center = getPointValue(optarg);
                    options.hasCenterOpt = true;
                    break;
                case 405:
                    if(!options.hasInverseCircleOpt) {
                          showError("No --inverse_circle flag for using
--radius flag", ARGUMENTS_FAILURE);
                    }
                    options.radius = getNonNegativeNumber(optarg);
                    options.hasRadiusOpt = true;
                    break;
                case 406:
                    if(!options.hasTrimOpt) {
                          showError("No --trim flag for using --left_up
flag", ARGUMENTS_FAILURE);
                    }
                    options.leftUp = getPointValue(optarg);
                    options.hasLeftUpOpt = true;
                    break;
                case 407:
                    if(!options.hasTrimOpt) {
                                showError("No --trim flag for using --
right_down flag", ARGUMENTS_FAILURE);
                    }
                    options.rightDown = getPointValue(optarg);
                    options.hasRightDownOpt = true;
                    break;
                case 408:
                    if(!options.hasTriangleOpt) {
                            showError("No --triangle flag for using --
points flag", ARGUMENTS_FAILURE);
                    }
                    options.points = getPointsValue(optarg);
                    options.hasPointsOpt = true;
                    break;
                case 409:
                     if(!options.hasTriangleOpt && !options.hasLineOpt)
{
```

```
                            showError("No --triangle or --line flag for
using --thickness flag", ARGUMENTS_FAILURE);
                    }
                    options.thickness = getNonNegativeNumber(optarg);
                    options.hasThicknessOpt = true;
                    break;
                case 410:
                     if(!options.hasTriangleOpt && !options.hasLineOpt)
{
                            showError("No --triangle or --line flag for
using --color flag", ARGUMENTS_FAILURE);
                    }
                    options.color = getRGBValue(optarg);
                    options.hasColorOpt = true;
                    break;
                case 411:
                    if(!options.hasTriangleOpt) {
                         showError("No --triangle flag for using --fill
flag", ARGUMENTS_FAILURE);
                    }
                    options.hasFillOpt = true;
                    break;
                case 412:
                    if(!options.hasTriangleOpt) {
                             showError("No --triangle flag for using --
fill_color flag", ARGUMENTS_FAILURE);
                    }
                    options.fillColor = getRGBValue(optarg);
                    options.hasFillColorOpt = true;
                    break;
                case 413:
                    if(!options.hasLineOpt) {
                            showError("No --line flag for using --start
flag", ARGUMENTS_FAILURE);
                    }
                    options.start = getPointValue(optarg);
                    options.hasStartOpt = true;
                    break;
                case 414:
```

```
                    if(!options.hasLineOpt) {
                            showError("No --line flag for using --end
flag", ARGUMENTS_FAILURE);
                    }
                    options.end = getPointValue(optarg);
                    options.hasEndOpt = true;
                    break;
                case 415:
                    options.hasInfoOpt = true;
                    break;
                default:
                        showError("Unknown option or missing argument",
ARGUMENTS_FAILURE);
            }
        }

        //Проверка на наличие функции
              if(!options.hasHelpOpt   &&   !options.hasInfoOpt   &&   !
options.hasInverseCircleOpt     &&     !options.hasTrimOpt     &&     !
options.hasTriangleOpt && !options.hasLineOpt) {
            showError("No function is selected!", ARGUMENTS_FAILURE);
        }

        if(options.hasHelpOpt) {
            printHelp();
            exit(EXIT_SUCCESS);
        }

        //Проверка на наличие флагов у inverse_circlea
         if(options.hasInverseCircleOpt && (!options.hasCenterOpt || !
options.hasRadiusOpt)) {
                showError("Not  enough  flags  in  --inverse_circle!",
ARGUMENTS_FAILURE);
        }

        //Проверка на наличие флагов у trim
             if(options.hasTrimOpt   &&   (!options.hasLeftUpOpt   ||   !
options.hasRightDownOpt)) {
```

```
                        showError("Not enough flags in --trim!",
ARGUMENTS_FAILURE);
        }

        //Проверка на наличие флагов у triangle
        if(options.hasTriangleOpt) {
            if(options.hasFillOpt && !options.hasFillColorOpt) {
                 showError("If there is a --fill flag, then it must be
--fill_color!", ARGUMENTS_FAILURE);
            }
              if(!options.hasPointsOpt || !options.hasThicknessOpt || !
options.hasColorOpt) {
                        showError("Not enough flags in --triangle!",
ARGUMENTS_FAILURE);
            }
        }

        //Проверка на наличие флагов у line
              if(options.hasLineOpt && (!options.hasStartOpt || !
options.hasEndOpt || !options.hasThicknessOpt || !options.hasColorOpt)) {
                        showError("Not enough flags in --line!",
ARGUMENTS_FAILURE);
        }

        //Проверка на наличие имени ыходного файла
        if(!options.hasInputOpt) {
            if(optind == argc - 1) {
                options.inputFileName = argv[argc - 1];
                options.hasInputOpt = true;
            }else if (optind < argc - 1) {
                showError("Too many arguments!", ARGUMENTS_FAILURE);
            } else {
                if(!options.hasHelpOpt) {
                       showError("There is no name of the input file!",
ARGUMENTS_FAILURE);
                }
            }
        }else {
            if (optind <= argc - 1) {
```

```
                showError("Too many arguments!", ARGUMENTS_FAILURE);
            }
        }

        //Проверка на схожесть имен входного и выходного файлов
        if(options.hasOutputOpt && options.hasInputOpt) {
              if (strcmp(options.inputFileName, options.outputFileName)
== 0) {
                  showError("Input and output files can't have the same
name!", ARGUMENTS_FAILURE);
            }
        }


        return options;
    }


    Point
    getPointValue(char* string) {
        int arr[2];
        int index = 0;
        Point new_point = {};

        if(string[0] == '.' || string[strlen(string) - 1] == '.') {
            showError("Input data error!", DATA_FAILURE);
        }

        char* token = strtok(string, ".");
        while(token != NULL && index < 2) {
            arr[index] = atoi(token);

            token = strtok(NULL, ".");
            index++;
        }

        if(token != NULL || index != 2) {
            showError("Input data error!", DATA_FAILURE);
        }
```

```
    new_point.x = arr[0];
    new_point.y = arr[1];

    return new_point;
}

RGB
getRGBValue(char* string){
    int arr[3];
    int index = 0;
    RGB new_color = {};

    if(string[0] == '.' || string[strlen(string) - 1] == '.') {
        showError("Input data error!", DATA_FAILURE);
    }

    char* token = strtok(string, ".");
    while(token != NULL && index < 3) {
        arr[index] = atoi(token);
        if(arr[index] > 255 || arr[index] < 0) {
            showError("Input data error!", DATA_FAILURE);
        }

        token = strtok(NULL, ".");
        index++;
    }

    if(token != NULL || index != 3) {
        showError("Input data error!", DATA_FAILURE);
    }

    new_color.r = arr[0];
    new_color.g = arr[1];
    new_color.b = arr[2];

    return new_color;
}
```

```
Point*
    getPointsValue(char* string) {
    if(string[0] == '.' || string[strlen(string) - 1] == '.') {
        showError("Input data error!", DATA_FAILURE);
    }

    int count = 0;
    int subCount = 0;
    Point* new_points = malloc(sizeof(Point) * 3);
      if(new_points == NULL) showError("Memory allocation error!",
MEMORY_ALLOCATION_FAILURE);;


    char* token = strtok(string, ".");

    while(token != NULL && count != 3) {
        if(subCount % 2 == 0) {
            new_points[count].x = atoi(token);
        }else {
            new_points[count].y = atoi(token);
            count++;
        }

        token = strtok(NULL, ".");
        subCount++;
    }

    if(token != NULL || count != 3) {
        showError("Input data error!", DATA_FAILURE);
    }

    return new_points;
}
int
getNonNegativeNumber(char* string) {
    int result = atoi(string);
    if (result <= 0) {
        showError("Input data error!", DATA_FAILURE);
```

```
        }

        return result;
    }

    void
    printHelp() {
            printf("Course work for option 5.5, created by Lapshov
Konstantin.\n");
        printf("Usage: ./programName [Flags] [InputFileName]\n\n");
        printf("Main flags:\n");
        printf("  -h, --help                                Display this
help message\n");
        printf("  --info                                  Print detailed
information about the input PNG file\n");
        printf("  -i, --input <filename>                    Specify the
name of the input BMP file\n");
        printf("  -o, --output <filename>                   Specify the
name of the output BMP file [default: out.png]\n");
        printf("  --inverse_circle [--center --radius]   Inverting the
colors in a given circle\n");
        printf("                                                Required
additional flags\n\n");
        printf("  --trim [--left_up --right_down]         Cropping the
image.\n");
        printf("                                                Required
additional flags\n\n");
         printf("  --triangle [--points --thickness --color --fill --
fill_color]   Drawing a triangle.\n");
                                                                printf("
Required additional flags\n\n");
        printf("  --line [--start --end --thickness --color]   Drawing
a line segment.\n");
        printf("                                                Required
additional flags\n\n");
        printf("Additional flags:\n");
         printf("  --center <x.y>                    Coordinates of the
center of the circle\n");
```

```
          printf("  --radius <value>                  The radius of the
circle\n");
          printf("  --left_up <x.y>                 Coordinates of the up
left corner of the image\n");
          printf("  --right_down <x.y>                Coordinates of the
bottom right corner of the image\n");
           printf("  --points <x1.y1.x2.y2.x3.y3>    Coordinates of the
vertices of the triangle\n");
          printf("  --thickness <value>             The thickness of the
line\n");
          printf("  --color <rrr.ggg.bbb>           The color of the line\
n");
          printf("  --fill                            Indicates whether a
fill is needed\n");
          printf("  --fill_color <rrr.ggg.bbb>      The color of the fill\
n");
          printf("  --start <x.y>                     Coordinates of the
start of the line\n");
          printf("  --end <x.y>                       Coordinates of the
end of the line\n\n");
          printf("Attention! If the --input flag indicating the name of
the input file is not specified, then after each main flag (except for
the --help and --info flags), at the end, you must specify the name of
the input file!\n\n");
     }
```

Файл bmpInteraction.c:

```
#include "bmpInteraction.h"


int
getOffset(size_t width){
    unsigned int offset = (width * sizeof(RGB)) % 4;
    offset = (offset ? 4-offset : 0);
    return offset;
}
```

```
BMP
readBMP(const char *filename) {
    FILE *bmpFile = fopen(filename, "rb");
    if (bmpFile == NULL) {
          showError("The file cannot be opened!\nCheck the name of
the input file", FILE_OPEN_FAILURE);
    }

    BMP bmp;
    fread(&bmp.bmfh, 1, sizeof(bmp.bmfh), bmpFile);
    fread(&bmp.bmih, 1, sizeof(bmp.bmih), bmpFile);
     if (bmp.bmfh.bfSign != 0x4D42 || bmp.bmih.biBitSize != 24 ||
bmp.bmih.biCompression != 0) {
        showError("Wrong file type!", FILE_READ_FAILURE);
    }

    const size_t height = bmp.bmih.biHeight;
    const size_t width = bmp.bmih.biWidth;

    // Перемещаем указатель файла на начало пиксельных данных
    fseek(bmpFile, bmp.bmfh.bfArrOffset, SEEK_SET);

    bmp.img = createArrayOfPixels(height, width);
    for (size_t i = 0; i < height; i++) {
                  fread(bmp.img[i],  1,  width  *  sizeof(RGB)  +
getOffset(width), bmpFile);
    }

    fclose(bmpFile);
    return bmp;
}


void
saveBMP(const char *filename, BMP bmp) {
    FILE *bmpFile = fopen(filename, "wb");
    if (bmpFile == NULL) {
        showError("The file cannot be saved!", FILE_WRITE_FAILURE);
    }
```

```
        fwrite(&bmp.bmfh, 1, sizeof(bmp.bmfh), bmpFile);
        fwrite(&bmp.bmih, 1, sizeof(bmp.bmih), bmpFile);

        const size_t height = bmp.bmih.biHeight;
        const size_t width = bmp.bmih.biWidth;

        // Перемещаем указатель файла на начало пиксельных данных
        fseek(bmpFile, bmp.bmfh.bfArrOffset, SEEK_SET);

        for (size_t i = 0; i < height; i++) {
                    fwrite(bmp.img[i],  1,  width  *  sizeof(RGB)  +
getOffset(width), bmpFile);
        }
        fclose(bmpFile);
    }

    RGB**
    createArrayOfPixels(int height, int width) {
        RGB **new_arr = (RGB **)malloc(height * sizeof(RGB *));
           if(new_arr  ==  NULL)  showError("Memory  allocation  error!",
MEMORY_ALLOCATION_FAILURE);
        for(int i = 0; i < height; i++) {
            new_arr[i] = (RGB *)malloc(width * sizeof(RGB));
                  if(new_arr[i]  ==  NULL)  showError("Memory  allocation
error!", MEMORY_ALLOCATION_FAILURE);
        }
        return new_arr;
    }

    void
    freeArrayOfPixels(BMP* bmp) {
        for (int i = 0; i < bmp->bmih.biHeight; i++) {
            free(bmp->img[i]);
        }
        free(bmp->img);
    }

    void
    printInfo(BMP bmp){
```

```
        printf("Signature:              \t%x (%hu)\n", bmp.bmfh.bfSign,
bmp.bmfh.bfSign);
         printf("Size of file:           \t%x (%u)\n", bmp.bmfh.bfSize,
bmp.bmfh.bfSize);
                printf("First   reserved   field:        \t%x   (%hu)\n",
bmp.bmfh.bfReserved1, bmp.bmfh.bfReserved1);
                 printf("Second   reserved   field:      \t%x   (%hu)\n",
bmp.bmfh.bfReserved2, bmp.bmfh.bfReserved2);
               printf("Pixel   array   offset:            \t%x   (%u)\n",
bmp.bmfh.bfArrOffset, bmp.bmfh.bfArrOffset);
         printf("Size of header:         \t%x (%u)\n", bmp.bmih.biSize,
bmp.bmih.biSize);
         printf("Image width:            \t%x (%u)\n", bmp.bmih.biWidth,
bmp.bmih.biWidth);
             printf("Image  height:                      \t%x  (%u)\n",
bmp.bmih.biHeight, bmp.bmih.biHeight);
            printf("Planes:                             \t%x  (%hu)\n",
bmp.bmih.biPlanes, bmp.bmih.biPlanes);
             printf("Bits  for  pixel:                  \t%x  (%hu)\n",
bmp.bmih.biBitSize, bmp.bmih.biBitSize);
             printf("Compression:                        \t%x  (%u)\n",
bmp.bmih.biCompression, bmp.bmih.biCompression);
             printf("Size  of  image:                    \t%x  (%u)\n",
bmp.bmih.biImageSize, bmp.bmih.biImageSize);
               printf("Pixels   per   meter   (x):        \t%x   (%u)\n",
bmp.bmih.biXPixelsPerMeter, bmp.bmih.biXPixelsPerMeter);
               printf("Pixels   per   meter   (y):        \t%x   (%u)\n",
bmp.bmih.biYPixelsPerMeter, bmp.bmih.biYPixelsPerMeter);
                printf("Colors   in   color   table:      \t%x   (%u)\n",
bmp.bmih.biClrTotal, bmp.bmih.biClrTotal);
                 printf("Important   color   count:      \t%x   (%u)\n",
bmp.bmih.biClrImportant, bmp.bmih.biClrImportant);
    }
```

Файл functIonsInteraction.c:

```
#include "functionsInteraction.h"
```

```
#define MAX(x, y) (((x) > (y)) ? (x) : (y))
#define MIN(x, y) (((x) < (y)) ? (x) : (y))

void
swapInt(int* first, int* second) {
    int temp = *first;
    *first = *second;
    *second = temp;
}
void
setPixel(BMP *bmp, Point point, RGB color){
    bmp->img[bmp->bmih.biHeight - point.y - 1][point.x] = color;
}
RGB
getInvertPixelColor(BMP *bmp, Point point) {
        RGB pixel = bmp->img[bmp->bmih.biHeight - point.y - 1]
[point.x];
        RGB invertPixel = {255 - pixel.b, 255 - pixel.g, 255 -
pixel.r};

    return invertPixel;
}



void
fillCircle(BMP *bmp, Point center, int radius, RGB color, bool
invert) {
    if(!invert) radius = (int)floor(radius/2);

    for(int x = -radius; x < radius; x++) {
         if (center.x + x > bmp->bmih.biWidth || center.x +x < 0)
continue;

          int height = sqrt(radius*radius - x*x);
          for (int y = -height; y < height; y++) {
              Point pointForFill = {x+center.x, y+center.y};
              if(pointInImage(bmp->bmih, pointForFill)) {
```

```
                        if (invert) color = getInvertPixelColor(bmp,
pointForFill);
                    setPixel(bmp, pointForFill, color);
                }
            }
        }
    }


    bool
    pointInImage(BitmapInfoHeader bmif, Point point) {
         if(point.x < 0 || point.y < 0 || point.x >= bmif.biWidth ||
point.y >= bmif.biHeight) {
            return false;
        }
        return true;
    }


    void
    line(BMP *bmp, Point start, Point end, int thickness, RGB color){
        int dx = abs(end.x - start.x);
        int dy = abs(end.y - start.y);
        int signX = (start.x < end.x) ? 1 : -1;
        int signY = (start.y < end.y) ? 1 : -1;
        int err = dx - dy;

        while (start.x != end.x || start.y != end.y) {
            if(thickness == 1) {
                if(pointInImage(bmp->bmih, start)) {
                    setPixel(bmp, start, color);
                }else {
                    break;
                }
            }else {
                fillCircle(bmp, start, thickness, color, false);
            }
            int err2 = err * 2;
```

```
            if(err2 > -dy) {
                err -= dy;
                start.x += signX;
            }
            if(err2 < dx) {
                err += dx;
                start.y += signY;
            }
        }
    }


    int
    checkCoordsInTriangle(Point cords, Point firstPoint, Point
secondPoint, Point thirdPoint) {
            int res1 = (firstPoint.x - cords.x) * (secondPoint.y-
firstPoint.y) - (secondPoint.x - firstPoint.x) * (firstPoint.y -
cords.y);
            int res2 = (secondPoint.x - cords.x) * (thirdPoint.y-
secondPoint.y) - (thirdPoint.x - secondPoint.x) * (secondPoint.y -
cords.y);
             int res3 = (thirdPoint.x - cords.x) * (firstPoint.y-
thirdPoint.y) - (firstPoint.x - thirdPoint.x) * (thirdPoint.y - cords.y);

         return (res1 > 0 && res2 > 0 && res3 > 0) || (res1 < 0 && res2
< 0 && res3 < 0);
    }


    void
    triangle(BMP *bmp, Point* points, int thickness, RGB color, int
fill, RGB fill_color) {
        Point firstPoint = points[0];
        Point secondPoint = points[1];
        Point thirdPoint = points[2];

        if(fill) {
```

```
                    for(int  i  =  MIN(firstPoint.y,  MIN(secondPoint.y,
thirdPoint.y)); i < MAX(firstPoint.y, MAX(secondPoint.y, thirdPoint.y));
i++) {
                      for(int  j  =  MIN(firstPoint.x,  MIN(secondPoint.x,
thirdPoint.x)); j < MAX(firstPoint.x, MAX(secondPoint.x, thirdPoint.x));
j++) {
                    Point pointForFill = {j, i};
                      if(checkCoordsInTriangle(pointForFill, firstPoint,
secondPoint, thirdPoint)) {
                                if(pointInImage(bmp->bmih, pointForFill))
setPixel(bmp, pointForFill, fill_color);
                    }
                }
            }
        }

        line(bmp, firstPoint, secondPoint, thickness, color);
        line(bmp, firstPoint, thirdPoint, thickness, color);
        line(bmp, secondPoint, thirdPoint, thickness, color);
    }


    void
    trim(BMP* bmp, Point leftUp, Point rightDown) {
      if(rightDown.x < leftUp.x) swapInt(&leftUp.x, &rightDown.x);
        if(leftUp.y > rightDown.y) swapInt(&leftUp.y, &rightDown.y);

        if(leftUp.x < 0) leftUp.x = 0;
        if(leftUp.y < 0) leftUp.y = 0;
            if(rightDown.x  >  bmp->bmih.biWidth)  rightDown.x  =  bmp-
>bmih.biWidth;
           if(rightDown.y  >  bmp->bmih.biHeight)  rightDown.y  =  bmp-
>bmih.biHeight;

        int new_height = rightDown.y - leftUp.y;
        int new_width = rightDown.x - leftUp.x;

        if(new_height <= 0 || new_width <= 0) {
            showError("Wrong coordinates!", ARGUMENTS_FAILURE);
```

```
        }

         // Создания нового массива пикселей с учетом предоставленных
данных
        RGB **new_arr = createArrayOfPixels(new_height, new_width);

        for(int i = leftUp.y; i < rightDown.y; i++) {
            for (int j = leftUp.x; j < rightDown.x; j++) {
                 new_arr[new_height - (i - leftUp.y) - 1][j - leftUp.x]
= bmp->img[bmp->bmih.biHeight - i - 1][j];
            }
        }

        //Очистка старого массива
        freeArrayOfPixels(bmp);

        //Работа со структурой файла с учетом новых данных
        bmp->bmih.biHeight = new_height;
        bmp->bmih.biWidth = new_width;

            bmp->bmih.biImageSize = 3 * new_width * new_height +
(new_height * (new_width % 4));
                bmp->bmfh.bfSize = bmp->bmfh.bfArrOffset + bmp-
>bmih.biImageSize;
        bmp->img = new_arr;
    }


    void
    choiceTask(BMP *bmp, Option option) {
        if(option.hasInfoOpt) {
            printInfo(*bmp);
            exit(EXIT_SUCCESS);
        }
        if(option.hasInverseCircleOpt) {
            RGB fakeColor = {0, 0, 0};
               fillCircle(bmp, option.center, option.radius, fakeColor,
true);
        }
```

```
        if(option.hasTrimOpt) {
            trim(bmp, option.leftUp, option.rightDown);
        }
        if(option.hasTriangleOpt) {
                        triangle(bmp,  option.points,  option.thickness,
option.color, option.hasFillOpt, option.fillColor);
            //Освобождение массива точек
            free(option.points);
        }
        if(option.hasLineOpt) {
                line(bmp,  option.start,  option.end,  option.thickness,
option.color);
        }
    }
```

Файл errors.c:

```
#include "errors.h"

void
showError(char* message, int errorType) {
    printf("%s\n", message);
    exit(errorType);
}
```

Файл errors.h:

```
#pragma once
#include <stdlib.h>
#include <stdio.h>

#define FILE_OPEN_FAILURE 40
#define FILE_READ_FAILURE 41
#define FILE_WRITE_FAILURE 42
#define MEMORY_ALLOCATION_FAILURE 43
#define ARGUMENTS_FAILURE 44
#define DATA_FAILURE 45

void showError(char* message, int errorType);
```

Файл structures.h:

```
#pragma once

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

#pragma pack(push, 1)
typedef struct {
    unsigned short bfSign;
    unsigned int bfSize;
    unsigned short bfReserved1;
    unsigned short bfReserved2;
    unsigned int bfArrOffset;
} BitmapFileHeader;

typedef struct {
    unsigned int biSize;
    unsigned int biWidth;
    unsigned int biHeight;
    unsigned short biPlanes;
    unsigned short biBitSize;
    unsigned int biCompression;
    unsigned int biImageSize;
    unsigned int biXPixelsPerMeter;
    unsigned int biYPixelsPerMeter;
    unsigned int biClrTotal;
    unsigned int biClrImportant;
} BitmapInfoHeader;

typedef struct {
    unsigned char b;
    unsigned char g;
    unsigned char r;
}RGB;
#pragma pack(pop)
```

```
typedef struct{
    BitmapFileHeader bmfh;
    BitmapInfoHeader bmih;
    RGB **img;
}BMP;

typedef struct {
    int x;
    int y;
}Point;

typedef struct {
    char* inputFileName;
    char* outputFileName;

    bool hasHelpOpt;
    bool hasInputOpt;
    bool hasOutputOpt;
    bool hasInverseCircleOpt;
    bool hasTrimOpt;
    bool hasTriangleOpt;
    bool hasLineOpt;
    bool hasCenterOpt;
    bool hasRadiusOpt;
    bool hasLeftUpOpt;
    bool hasRightDownOpt;
    bool hasPointsOpt;
    bool hasThicknessOpt;
    bool hasColorOpt;
    bool hasFillOpt;
    bool hasFillColorOpt;
    bool hasStartOpt;
    bool hasEndOpt;
    bool hasInfoOpt;

    int radius;
    int thickness;

    Point center;
```

```
    Point leftUp;
    Point rightDown;
    Point* points;
    Point start;
    Point end;

    RGB color;
    RGB fillColor;
}Option;
```

Файл cliInterface.h:

```
#pragma once
#include <getopt.h>
#include "structures.h"
#include "errors.h"

Option getCliFlagsInfo(int argc, char *argv[]);

Point getPointValue(char* string);
Point* getPointsValue(char* string);
RGB getRGBValue(char* string);
int getNonNegativeNumber(char* string);
void printHelp();
```

Файл bmpInteraction.h:

```
#pragma once

#include "structures.h"
#include "errors.h"

int getOffset(size_t width);
BMP readBMP(const char *filename);
void saveBMP(const char *filename, BMP bmp);
RGB** createArrayOfPixels(int height, int width);
void freeArrayOfPixels(BMP* bmp);
```

```
void printInfo(BMP header);
```

Файл functionsInteraction.h:

```
#pragma once

#include <math.h>
#include "structures.h"
#include "errors.h"
#include "bmpInteraction.h"

void swapInt(int* first, int* second);
void setPixel(BMP *bmp, Point point, RGB color);
bool pointInImage(BitmapInfoHeader bmif, Point point);
RGB getInvertPixelColor(BMP *bmp, Point point);

void fillCircle(BMP *bmp, Point center, int radius, RGB color, bool
invert);
void trim(BMP* bmp, Point leftUp, Point rightDown);
void line(BMP *bmp, Point start, Point end, int thickness, RGB
color);
void triangle(BMP *bmp, Point* points, int thickness, RGB color,
int fill, RGB fill_color);
void choiceTask(BMP *bmp, Option option);
```

Файл Makefile:

```
CC = gcc
CFLAGS = -c -Wall -std=c99

all: main.o bmpInteraction.o cliInterface.o functionsInteraction.o
errors.o
	$(CC) main.o bmpInteraction.o cliInterface.o
functionsInteraction.o errors.o -o cw -lm

main.o: main.c structures.h bmpInteraction.h cliInterface.h
functionsInteraction.h
	$(CC) $(CFLAGS) main.c
```

```
bmpInteraction.o: bmpInteraction.c bmpInteraction.h structures.h
errors.h
	$(CC) $(CFLAGS) bmpInteraction.c

cliInterface.o: cliInterface.c cliInterface.h structures.h errors.h
	$(CC) $(CFLAGS) cliInterface.c

functionsInteraction.o: functionsInteraction.c
functionsInteraction.h structures.h errors.h bmpInteraction.h
	$(CC) $(CFLAGS) functionsInteraction.c

errors.o: errors.c errors.h
	$(CC) $(CFLAGS) errors.c

clean:
		rm *.o
```