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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка изображений

Студент гр. 3341 _____ Бойцов В.А.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2024

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Бойцов Владислав

Группа 3341

Тема работы: Обработка изображений

Вариант 4.3

Программа **обязательно должна иметь CLI** (опционально дополнительное использование GUI). Более подробно тут: **http://se.moevm.info/doku.php/courses:programming:rules_extra_kurs**

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 прямоугольника. Флаг для выполнения данной операции: `--rect`. Он определяется:

- Координатами левого верхнего угла. Флаг `--left_up`, значение задаётся в формате `left.up`, где left – координата по x, up – координата по y
- Координатами правого нижнего угла. Флаг `--right_down`, значение задаётся в формате `right.down`, где right – координата по x, down – координата по y
- 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)
- Прямоугольник может быть залит или нет. 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) Сделать рамку в виде узора. Флаг для выполнения данной операции: `--ornament`. Рамка определяется:
  - Узором. Флаг `--pattern`. Обязательные значения: rectangle и circle, semicircles. Также можно добавить свои узоры (красивый узор можно получить используя фракталы). Подробнее здесь: https://se.moevm.info/doku.php/courses:programming:cw_spring_ornament
  - 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)
  - Шириной. Флаг `--thickness`. На вход принимает число больше 0
  - Количеством. Флаг `--count`. На вход принимает число больше 0
  - При необходимости можно добавить дополнительные флаги

для необозначенных узоров

- (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.

Содержание пояснительной записки:

Аннотация, содержание, введение, ход выполнения, заключение, приложения.

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 20.05.2024

Дата защиты реферата: 22.05.2024

Студент _____ Бойцов В.А.

Преподаватель _____ Глазунов С.А.

АННОТАЦИЯ

Курсовая работа по варианту 4.3 представляет собой программу на языке C++, обрабатывающую изображения формата BMP.

Программа принимает на вход флаги из командной строки, обрабатывает их и выполняет определенные функции с изображением, затем сохраняет его изменённую версию, если поданные аргументы были корректными.

Демонстрация работы программы представлена на рис. 1 и рис.2 со следующими входными данными: *./cw --ornament --pattern semicircles --color 50.6.100 --count 10 --thickness 5 --output ../samples/annotation.bmp ../samples/sample2.bmp*

Рисунок 1 – входное изображение

Рисунок 2 – пример работы программы

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью курсовой работы является изучение формата файлов BMP, а также реализация функций для работы с этими форматами файлов.

Для достижения поставленной цели требуется решить следующие задачи:

1. Изучить BMP формат изображений;
2. Получить информацию об изображении: размеры, содержимое и др.;
3. Обработать массив пикселей в соответствии с заданием;
4. Обработать исключительные случаи;
5. Сохранить итоговое изображение в новый файл.

1. ХОД ВЫПОЛНЕНИЯ РАБОТЫ

1.1. Флаги, их считывание и обработка

Пользователь взаимодействует с программой посредством аргументов командной строки. Для их считывания и обработки используется функционал библиотеки *getopt.h*. Массив структур *const struct option long_options[]* хранит в себе подробную информацию о каждом флаге, с которые должна обрабатывать программа, а строка *const char* short_options = "hi:o:"* – информацию о всех возможных коротких версиях этих флагов. Для прочтения и обработки флагов используется функция библиотечная *getopt_long()*.

Для вызова этой функции и взаимодействия с пользователем реализован класс *UserInteraction*. Его метод *void readFlags(int argc, char** argv)* с помощью *getopt_long()* считывает флаги и их аргументы, поступившие из командной строки, и записывает их в статический член *класса std::map<int, std::string> flagValues*. Также этот класс содержит методы для парсинга аргументов разных типов, для взаимодействия с пользователем.

Данные, добытые в *UserInteraction*, передаются в объект класса *DataStorage*. Там, посредством методов из *UserInteraction*, эти аргументы парсятся, а затем вызываются соответствующие требуемой задаче методы графического модуля.

1.2. Хранение, чтение и запись изображения

Для хранения данных об изображении, а также самого массива пикселей изображения, созданы структуры *BmpFileHeader* и *BmpInfoHeader*, хранящие информацию о заголовке изображения, и *Rgb*, хранящая информацию о пикселе изображения для синей, зеленой и красной компонент.

В программе для работы с изображениями реализован класс *BmpImage*, содержащий в себе поля *BmpFileHeader mBmfh* и *BmpInfoHeader mBmih* для хранения заголовка изображения, *mHeight u mWidth* для хранения его размеров, *std::string filePath* для хранения пути, по которому оно содержится в памяти компьютера, и *Rgb** mPixelArr* - массива его пикселей.

Для считывания изображения реализованы конструкторы класса *BmpImage()*, по-разному создающие объект класса в зависимости от переданных параметров, и метод *open()*, открывающий изображение для уже созданного объекта. В них – если передан корректный адрес изображения – считываются, соблюдая смещение, данные файла с помощью *fread()*.

Выгрузка изображения в файл реализуется с помощью метода *save()*, в котором данные полей класса загружаются в файл с помощью *fwrite()*.

1.3. Обработка изображения

Для изменения изображения были реализованы многочисленные методы, позволяющие решать многие задачи. Для поставленных задач используется:

- *drawSmoothRectangle()* для рисования прямоугольника. Метод попиксельно закрашивает область внутри прямоугольника, если есть заливка, затем рисует границы прямоугольника с помощью функции *drawLine()*. *DrawLine()* работает по алгоритму Брезенхема, а толщина линий достигается за счёт рисования кругов методом *drawCircle()*;
- один из методов *circleOrnament(), rectangleOrnament()* или *semiCircleOrnament()*– в зависимости от узора рамки – для рисования рамки изображения. Для рисования окружностей и прямоугольников используются методы *drawCircle()* и *drawRectangle()* (последний рисует прямоугольник без скругленных углов);
- один из методов *rotate180()*, *rotate90()* или *rotate270()*– в зависимости от угла поворота – для поворота части изображения. С помощью метода *copy()* копируется выделенная пользователем часть изображения (с учетом размеров исходного изображения), затем создаётся новый массив пикселей, куда записываются пиксели старого в необходимом для поворота порядке, а затем с помощью *paste()* вставляется в исходное изображение.

1.4. Makefile

Программа разбита на несколько файлов с исходным кодом и заголовочных файлов, а именно: *UserInteraction.cpp* (обработка флагов и взаимодействие с пользователем), *main.cpp*, *DataStorage.cpp* (хранилище временных данных и вызов функций обработки изображения), *BmpImage.cpp* (класс и для работы с изображениями и его методы), *OptionParsingConstants.cpp* (константы, необходимые для парсинга) и соответствующие им заголовочные файлы (не считая *main.c*; дополнительно для определения структур создан заголовочный файл *bmpStructures.h*).

Для компиляции программы используется *Makefile*. В нем определены пути к файлам с исходным кодом (*SRCDIR*), папке для объектных файлов (*BIULDDIR*), флаги для компилятора (*CXX*, т.е. *g++*).

Определяются все файлы с исходным кодом и соответствующие им объектные файлы.

Цель *all* мейкфайл’а вызывает цель *TARGET*, которая создаёт папку для объектных файлов, а затем из исходных файлов компилирует программу (исполняемый файл называется *cw*).

Цель clean удаляет исполняемый файл *cw* и все объектные файлы из папки *bin*.

Ознакомиться с примерами работы программы можно в приложении А, а с исходным кодом программы – в приложении В.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы был изучен формат файлов BMP, были реализованы функции для работы с ним.

Были решены поставленные задачи, разработаны способы считывать изображение, получать информацию о нём, считывать инструкции из командной строки и обрабатывать их. Были созданы функции, позволяющие изменять изображение в соответствии с введенными данными и обрабатывать исключительные случаи.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. getopt(3) – Linux manual page. URL: https://man7.org/linux/man-pages/man3/getopt.3.html

2. Image File Formats – BMP. URL: https://docs.fileformat.com/image/bmp/

3. Image File Formats – BMP. URL: https://docs.fileformat.com/image/bmp/

4. Bresenham J. E. Algorithm for computer control of a digital plotter // IBM Systems journal. 1965. Т. 4, №. 1. P. 25–30

ПРИЛОЖЕНИЕ А
ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

Все приведенные ниже примеры работы программы изменяют изображение, представленное на рис. 1.

Рисование прямоугольника с заливкой представлено на рис. 3, входные данные: *./cw --rect --left_up 50.51 --right_down 400.400 --color 50.10.100 --thickness 6 --fill --fill_color 0.0.200 --output ../samples/rect.bmp ../samples/sample2.bmp*

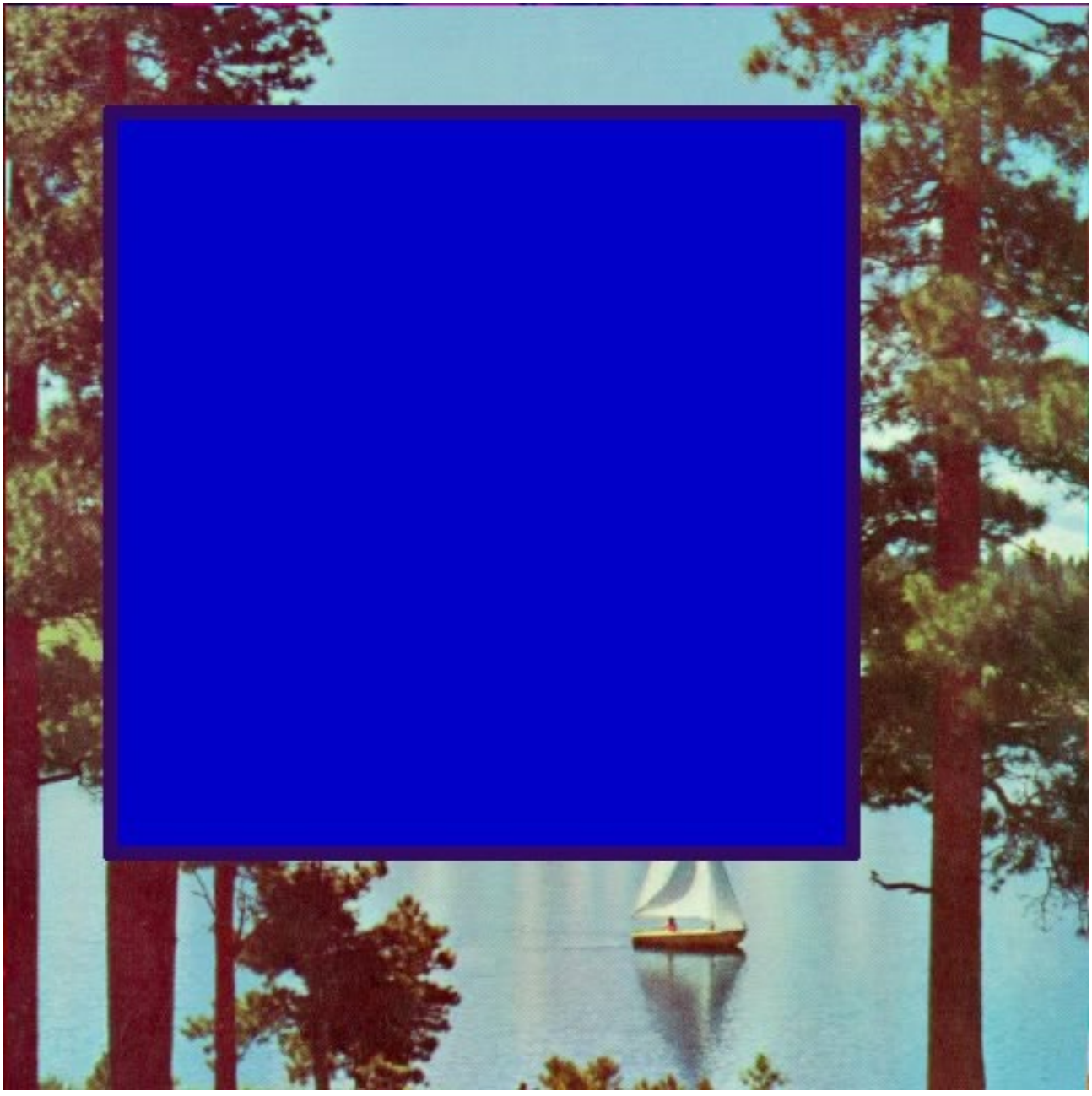

Рисунок 3 – рисование прямоугольника с заливкой

Пример рисования узора представлен на рис. 4, входные данные: *./cw --ornament --pattern circle --color 60.50.10 --output ../samples/annotation.bmp ../samples/sample2.bmp*

Рисунок 4 – рисование узора “circle”

Пример поворота области изображения проиллюстрирован на рис. 5, входные данные: *./cw --ornament --pattern circle --color 60.50.10 --output ../samples/annotation.bmp ../samples/sample2.bmp*

Рисунок 5 – поворот области изображения на 90 градусов

Вывод информации об изображении при команде *./cw --info ../samples/sample2.bmp*:

Course work for option 4.3, created by Vladislav Boitsov

Signature: 4d42 (19778)

Filesize: c0036 (786486)

Reserved 1: 0 (0)

Reserved 2: 0 (0)

PixelArrOffset: 36 (54)

Header Size: 28 (40)

Width: 200 (512)

Height: 200 (512)

Planes: 1 (1)

Bits per Pixel: 18 (24)

Compression: 0 (0)

Image Size: 0 (0)

X Pixels per Meter: b12 (2834)

Y Pixels per Meter: b12 (2834)

Colors in Color Table: 0 (0)

Important Color Number: 0 (0)

Обработка ошибок при неправитльных входных данных *./cw --rect --left_up 50.51 --right_down 400.400 --color 50.10.100 --thickness -200 --fill --fill_color 0.0.200 --output ../samples/rect.bmp ../samples/sample2.bmp*:

Course work for option 4.3, created by Vladislav Boitsov

Non-positive thickness!

Пример обработки граничных случаев проиллюстрирован на рис. 6, входные данные: *./cw --rotate --angle 90 --left_up 400.400 --right_down 400.400 --output ../samples/annotation.bmp ../samples/sample2.bmp*

Рисунок 6 – граничный случай для области поворота

ПРИЛОЖЕНИЕ В
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл: bmpStructures.h

```
#ifndef BMP_STRUCTURES_H
#define BMP_STRUCTURES_H

#define SIGNATURE_VALUE 0x4d42
#define BITS_PER_PIXEL_VALUE 24
#define COMPRESSION_VALUE 0

#pragma pack(push, 1)

typedef struct
{
    unsigned short signature;
    unsigned int fileSize;
    unsigned short reserved1;
    unsigned short reserved2;
    unsigned int pixelArrOffset;
} BmpFileHeader;

typedef struct
{
    unsigned int headerSize;
    unsigned int width;
    unsigned int height;
    unsigned short planes;
    unsigned short bitsPerPixel;
    unsigned int compression;
    unsigned int imageSize;
    unsigned int xPixelsPerMeter;
    unsigned int yPixelsPerMeter;
    unsigned int colorsInColorTable;
    unsigned int importantColorCount;
} BmpInfoHeader;
```

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

#pragma pack(pop)

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

#endif
```

Файл: UserInteraction.h

```
#ifndef USER_INTERACTION_H
#define USER_INTERACTION_H

#include <map>
#include <set>
#include <string>
#include <getopt.h>
#include "bmpStructures.h"
#include "OptionParsingConsts.h"

extern const struct option long_options[];
extern const char* short_options;

class UserInteraction
{
private:
    static inline std::map<int, std::string> flagValues={};

public:
    static std::string getInputPath();
    static std::string getOutputPath();
    static int getOption();
    static std::string getFlagValue(int flagIndex);
    static void readFlags(int argc, char** argv);
    static void checkRequiredFlags(std::set<int> requiredArgs);
```

```
        static void checkExtraFlags(std::set<int> argsList);
        static int checkReceivedFlag(int flagIndex);
        static int parseIntValue(int optind);
        static Rgb parseColorValue(int optind);
        static Point parseCoordValue(int optind);
        static void printHelp();
        static void printDevInfo();
        static  void  printImageInfo(BmpFileHeader  bmfh,  BmpInfoHeader
bmih);
    };

    #define ARG_FORMAT_ERROR 40
    #define IMG_FILE_ERROR 41
    #define ARG_VALUE_ERROR 42
    #define UNKNOWN_FLAG_ERROR 43
    #define MISSING_ARGS_ERROR 44
    #define MULTIPLE_ARGS_ERROR 45

    void throwInvalidFormat(std::string message);
    void throwUnknownFlag(std::string message);
    void throwMissingArgs(std::string message);
    void throwMultipleArgsError(std::string message);

    class InvalidImageException: public std::exception
    {
    private:
        std::string message;
    public:
        InvalidImageException(std::string message): message{message}{}
        const char* what() const noexcept override;
    };

#endif
```

Файл: UserInteraction.cpp

```
    #include "UserInteraction.h"
    #include <set>
    #include <map>
    #include <iostream>

    int UserInteraction::parseIntValue(int optind)
    {
        int value;
        if(sscanf(flagValues[optind].c_str(), "%d", &value)!=1)
            throwInvalidFormat("Invalid int format!");
```

```
        return value;
    }

    Rgb UserInteraction::parseColorValue(int optind)
    {
        Rgb color;
        int r, g, b;
        if(sscanf(flagValues[optind].c_str(), "%d.%d.%d", &(r), &(g),
&(b))!=3)
            throwInvalidFormat("Invalid Color format!");
        color.r=r;
        color.g=g;
        color.b=b;
        return color;
    }

    Point UserInteraction::parseCoordValue(int optind)
    {
        Point coord;
        if(sscanf(flagValues[optind].c_str(), "%d.%d", &(coord.x),
&(coord.y))!=2)
            throwInvalidFormat("Invalid Coords format!");
        return coord;
    }

    void UserInteraction::readFlags(int argc, char** argv)
    {
        int rez;
        int option_index;
        while((rez=getopt_long(argc, argv, short_options, long_options,
&option_index))!=-1)
        {
            if(flagValues.count(rez)==0)
            {
                if(rez=='?')
                {
                    throwUnknownFlag("Unknown option!");
                }
                else
                {
                    if(optarg)
                    {
                        std::string arg(optarg);
                        flagValues[rez]=arg;
```

```
                }
                else
                    flagValues[rez]="";
            }
        }
    }
    if(argc==1)
    {
        flagValues['h']="";
    }
    else if(flagValues.count('i')==0)
    {
        std::string arg(argv[argc-1]);
        flagValues['i']=arg;
    }
}

void              UserInteraction::checkRequiredFlags(std::set<int>
requiredArgs)
{
    for (auto i : requiredArgs)
        if(flagValues.count(i)==0)
            throwMissingArgs("Some required arguments missed!");
}

void UserInteraction::checkExtraFlags(std::set<int> argsList)
{
    for(auto i: flagValues)
        if(argsList.count(i.first)==0)
        {
            std::cout<<"Some args will be ignored!"<<"\n";
            break;
        }
}

std::string UserInteraction::getInputPath()
{
    if(flagValues.count('i')==0)
        throwMissingArgs("No input path received!");
    return flagValues['i'];
}

std::string UserInteraction::getOutputPath()
{
```

```
        if(flagValues.count('o')==0)
            throwMissingArgs("No output path received!");
        return flagValues['o'];
    }

    int UserInteraction::getOption()
    {
        std::set<int> mode;
        if(checkReceivedFlag(HELP_INDEX))
            mode.insert(HELP_INDEX);
        if(checkReceivedFlag(INFO_INDEX))
            mode.insert(INFO_INDEX);
        if(checkReceivedFlag(RECT_INDEX))
            mode.insert(RECT_INDEX);
        if(checkReceivedFlag(ORNAMENT_INDEX))
            mode.insert(ORNAMENT_INDEX);
        if(checkReceivedFlag(ROTATE_INDEX))
             mode.insert(ROTATE_INDEX);
        if(checkReceivedFlag(RHOMBUS_INDEX))
            mode.insert(RHOMBUS_INDEX);
        if(mode.size()==0)
            throwMissingArgs("No valid option received!");
        else if(mode.size()>1)
            throwMultipleArgsError("Too many options received!");
        return *(mode.begin());
    }

    int UserInteraction::checkReceivedFlag(int flagIndex)
    {
        return flagValues.count(flagIndex)!=0;
    }

    std::string UserInteraction::getFlagValue(int flagIndex)
    {
        return flagValues[flagIndex];
    }

    void UserInteraction::printDevInfo()
    {
        std::cout<<"Course work for option 4.3, created by Vladislav
Boitsov"<<std::endl;
    }
```

```
     void      UserInteraction::printImageInfo(BmpFileHeader      bmfh,
BmpInfoHeader bmih)
     {
          std::cout<<"Signature:\t"<<std::hex<<bmfh.signature<<"
("<<std::dec<<bmfh.signature<<")\n";
          std::cout<<"Filesize:\t"<<std::hex<<bmfh.fileSize<<"
("<<std::dec<<bmfh.fileSize<<")\n";
          std::cout<<"Reserved          1:\t"<<std::hex<<bmfh.reserved1<<"
("<<std::dec<<bmfh.reserved1<<")\n";
          std::cout<<"Reserved          2:\t"<<std::hex<<bmfh.reserved2<<"
("<<std::dec<<bmfh.reserved2<<")\n";

std::cout<<"PixelArrOffset:\t"<<std::hex<<bmfh.pixelArrOffset<<"
("<<std::dec<<bmfh.pixelArrOffset<<")\n\n";
          std::cout<<"Header        Size:\t"<<std::hex<<bmih.headerSize<<"
("<<std::dec<<bmih.headerSize<<")\n";
          std::cout<<"Width:\t\t"<<std::hex<<bmih.width<<"
("<<std::dec<<bmih.width<<")\n";
          std::cout<<"Height:\t\t"<<std::hex<<bmih.height<<"
("<<std::dec<<bmih.height<<")\n";
          std::cout<<"Planes:\t\t"<<std::hex<<bmih.planes<<"
("<<std::dec<<bmih.planes<<")\n";
          std::cout<<"Bits  per  Pixel:\t"<<std::hex<<bmih.bitsPerPixel<<"
("<<std::dec<<bmih.bitsPerPixel<<")\n";
          std::cout<<"Compression:\t"<<std::hex<<bmih.compression<<"
("<<std::dec<<bmih.compression<<")\n";
          std::cout<<"Image          Size:\t"<<std::hex<<bmih.imageSize<<"
("<<std::dec<<bmih.imageSize<<")\n";
          std::cout<<"X                      Pixels                    per
Meter:\t"<<std::hex<<bmih.xPixelsPerMeter<<"
("<<std::dec<<bmih.xPixelsPerMeter<<")\n";
          std::cout<<"Y                      Pixels                    per
Meter:\t"<<std::hex<<bmih.yPixelsPerMeter<<"
("<<std::dec<<bmih.yPixelsPerMeter<<")\n";
          std::cout<<"Colors                    in                   Color
Table:\t"<<std::hex<<bmih.colorsInColorTable<<"
("<<std::dec<<bmih.colorsInColorTable<<")\n";
          std::cout<<"Important                                      Color
Number:\t"<<std::hex<<bmih.importantColorCount<<"
("<<std::dec<<bmih.importantColorCount<<")\n"<<std::endl;
     }

     void UserInteraction::printHelp()
     {
```

```
        std::cout<<"\nHello!  I'm  Andrea,  your  personal  assistent
UwU!\n";
        std::cout<<"Let  me  tell  you,  how  to  handle  with  this
program)\n";
        std::cout<<"For proper functioning, you need to print proper
flags!\n";
        std::cout<<"For  all  of  them  except  of  'help',  which  is
obviously me,\n";
        std::cout<<"You will need to set a path to the image you want
to work with!\n";
        std::cout<<"For this purpose, use -i or --input or simply print
path as last arument!\n\n";
        std::cout<<"Today we have these options:\n\n";
        std::cout<<"1)  Getting  some  info  about  an  image  with  --
info\n\n";
        std::cout<<"For next options you will need to set a path for
saving the image!\n";
        std::cout<<"Use -o or --output to do this\n\n";
        std::cout<<"2) Drawing a rectangle with --rect. Use:\n";
         std::cout<<"\t--left_up and --right_down to set points of the
rectangle in 'x.y' format\n";
          std::cout<<"\t--thickness to set a thickness of the border\n";
          std::cout<<"\t--color  to  set  border  color  in  'rrr.ggg.bbb'
format\n";
          std::cout<<"\t--fill to set if to fill rectangle\n";
          std::cout<<"\t--fill_color  to  set  the  filling  color  in
'rrr.ggg.bbb' format\n\n";
        std::cout<<"3) Drawing an ornament with --ornament. Use:\n";
          std::cout<<"\t--pattent to set one of three patterns:\n";
          std::cout<<"\t'circle'. Use:\n";
          std::cout<<"\t\t--color  to  set  the  outer  color  in
'rrr.ggg.bbb' format\n";
          std::cout<<"\t'rectangle'. Use:\n";
          std::cout<<"\t\t--color to set the color of the ornament in
'rrr.ggg.bbb' format\n";
          std::cout<<"\t\t--thickness to set the borders thickness\n";
          std::cout<<"\t\t--count to set how many borders you want to
draw\n";
          std::cout<<"\t'semicircles'. Use:\n";
          std::cout<<"\t\t--color to set the color of the ornament in
'rrr.ggg.bbb' format\n";
          std::cout<<"\t\t--thickness  to  set  the  semicircles
thickness\n";
```

```
          std::cout<<"\t\t--count to set how many semicircles you want
to draw\n\n";
         std::cout<<"4) Rotating a part of an image with --rotate.
Use:\n";
          std::cout<<"\t--left_up and --right_down to set points of the
rotation zone in 'x.y' format\n";
          std::cout<<"\t--angle to set the rotation angle. Today we
support only 90, 180 and 270 deg. rotation\n\n";
         std::cout<<"Still have any questions? Use -h or --help to call
me!\n\n";
         std::cout<<"Have a nice day)\n"<<std::endl;
     }

     void throwInvalidFormat(std::string message)
     {
         std::cout<<message<<std::endl;
         exit(ARG_FORMAT_ERROR);
     }

     void throwUnknownFlag(std::string message)
     {
         std::cout<<message<<std::endl;
         exit(UNKNOWN_FLAG_ERROR);
     }

     void throwMissingArgs(std::string message)
     {
         std::cout<<message<<std::endl;
         exit(MISSING_ARGS_ERROR);
     }

     void throwMultipleArgsError(std::string message)
     {
         std::cout<<message<<std::endl;
         exit(MULTIPLE_ARGS_ERROR);
     }

     const char* InvalidImageException::what() const noexcept
     {
         return message.c_str();
}
```

Файл: DataStorage.h

```
#ifndef DATA_STORAGE_H
```

```
#define DATA_STORAGE_H

#include <string>
#include "BmpImage.h"
#include "UserInteraction.h"

class DataStorage
{
private:
    std::string mInputImagePath;
    std::string mOutputImagePath;
    int mOption;
    BmpImage mImage;

    void execInfo();
    void execRect();
    void execOrnament();
    void execRotate();
    void execRhombus();
public:
    DataStorage();
    void openImage();
    void executeOption();
    void saveImage();
};

#endif
```

Файл: DataStorage.cpp

```
#include "DataStorage.h"
#include "OptionParsingConsts.h"

DataStorage::DataStorage()
{
    BmpImage mImage;
    mOption=UserInteraction::getOption();
    if(mOption!=HELP_INDEX)
    {
        mInputImagePath=UserInteraction::getInputPath();
        if(mOption!=INFO_INDEX)
            mOutputImagePath=UserInteraction::getOutputPath();
    }
}
```

```
void DataStorage::openImage()
{
    try
    {
        mImage.open(mInputImagePath);
    }
    catch(const InvalidImageException& excp)
    {
        std::cout << excp.what() << '\n';
        exit(IMG_FILE_ERROR);
    }
}

void DataStorage::saveImage()
{
    try
    {
        mImage.save(mOutputImagePath);
    }
    catch(const InvalidImageException& excp)
    {
        std::cout << excp.what() << '\n';
        exit(IMG_FILE_ERROR);
    }
}

void DataStorage::executeOption()
{
    if(mOption!=HELP_INDEX)
        openImage();
    switch(mOption)
    {
        case HELP_INDEX:
        {
            UserInteraction::printHelp();
            break;
        };
        case INFO_INDEX:
        {
            DataStorage::execInfo();
            break;
        };
        case RECT_INDEX:
        {
```

```
                DataStorage::execRect();
                break;
            };
            case ORNAMENT_INDEX:
            {
                DataStorage::execOrnament();
                break;
            };
            case ROTATE_INDEX:
            {
                DataStorage::execRotate();
                break;
            };
            case RHOMBUS_INDEX:
            {
                DataStorage::execRhombus();
                break;
            };
        };
        if(mOption!=HELP_INDEX && mOption!=INFO_INDEX)
            saveImage();
    }

    void DataStorage::execInfo()
    {
        BmpFileHeader bmfh = mImage.getFileHeader();
        BmpInfoHeader bmih = mImage.getInfoHeader();
        UserInteraction::printImageInfo(bmfh, bmih);
    }

    void DataStorage::execRect()
    {
        Point leftUp, rightDown;
        int thickness;
        Rgb lineColor, fillColor={0, 0, 0};
        bool isFill=false;
        UserInteraction::checkRequiredFlags({LEFT_UP_INDEX,
RIGHT_DOWN_INDEX, THICKNESS_INDEX, COLOR_INDEX});
        UserInteraction::checkExtraFlags({RECT_INDEX,     LEFT_UP_INDEX,
RIGHT_DOWN_INDEX,      THICKNESS_INDEX,      FILL_INDEX,      COLOR_INDEX,
FILL_COLOR_INDEX, INPUT_INDEX, OUTPUT_INDEX});
        leftUp=UserInteraction::parseCoordValue(LEFT_UP_INDEX);
        rightDown=UserInteraction::parseCoordValue(RIGHT_DOWN_INDEX);
        thickness=UserInteraction::parseIntValue(THICKNESS_INDEX);
```

```
        lineColor=UserInteraction::parseColorValue(COLOR_INDEX);
        if(UserInteraction::checkReceivedFlag(FILL_INDEX)!=0)
        {
            isFill=true;
            UserInteraction::checkRequiredFlags({FILL_COLOR_INDEX});

fillColor=UserInteraction::parseColorValue(FILL_COLOR_INDEX);
        }
        try
        {
            mImage.drawRectangle(leftUp,      rightDown,      thickness,
lineColor, isFill, fillColor, true);
        }
        catch(const std::invalid_argument& excp)
        {
            std::cout << excp.what() << '\n';
            exit(ARG_VALUE_ERROR);
        }
    }

    void DataStorage::execOrnament()
    {
        UserInteraction::checkRequiredFlags({PATTERN_INDEX});
        UserInteraction::checkExtraFlags({ORNAMENT_INDEX,
PATTERN_INDEX,  THICKNESS_INDEX,  COUNT_INDEX,  COLOR_INDEX,  INPUT_INDEX,
OUTPUT_INDEX});
        std::string                        pattern                        =
UserInteraction::getFlagValue(PATTERN_INDEX);
        if(pattern == "rectangle" || pattern == "semicircles")
        {
            UserInteraction::checkRequiredFlags({THICKNESS_INDEX,
COUNT_INDEX, COLOR_INDEX});
            int
thickness=UserInteraction::parseIntValue(THICKNESS_INDEX);
            int count=UserInteraction::parseIntValue(COUNT_INDEX);
            Rgb color=UserInteraction::parseColorValue(COLOR_INDEX);
            try
            {
                if(pattern=="rectangle")
                    mImage.rectangleOrnament(color, thickness, count);
                else
                    mImage.semiCircleOrnament(color, thickness, count);
            }
            catch(const std::invalid_argument& excp)
```

```
            {
                std::cerr << excp.what() << '\n';
                exit(ARG_VALUE_ERROR);
            }
        }
        else if(pattern == "circle")
        {
            UserInteraction::checkRequiredFlags({COLOR_INDEX});
            Rgb color=UserInteraction::parseColorValue(COLOR_INDEX);
            try
            {
                mImage.circleOrnament(color);
            }
            catch(const std::invalid_argument& excp)
            {
                std::cerr << excp.what() << '\n';
                exit(ARG_VALUE_ERROR);
            }
        }
        else
            throwUnknownFlag("Unknown ornament pattern!");
    }

    void DataStorage::execRotate()
    {
        Point leftUp, rightDown;
        int angle;
        UserInteraction::checkRequiredFlags({LEFT_UP_INDEX,
RIGHT_DOWN_INDEX, ANGLE_INDEX});
        UserInteraction::checkExtraFlags({ROTATE_INDEX,  LEFT_UP_INDEX,
RIGHT_DOWN_INDEX, ANGLE_INDEX, INPUT_INDEX, OUTPUT_INDEX});
        leftUp=UserInteraction::parseCoordValue(LEFT_UP_INDEX);
        rightDown=UserInteraction::parseCoordValue(RIGHT_DOWN_INDEX);
        angle=UserInteraction::parseIntValue(ANGLE_INDEX);
        try
        {
            mImage.rotate(leftUp, rightDown, angle);
        }
        catch(const std::invalid_argument& excp)
        {
            std::cerr << excp.what() << '\n';
            exit(ARG_VALUE_ERROR);
        }
    }
```

```
void DataStorage::execRhombus()
{
    Point upper_vertex;
    int size=0;
    Rgb fill_color;
    UserInteraction::checkRequiredFlags({UPPER_VERTEX_INDEX,
SIZE_INDEX, FILL_COLOR_INDEX});
    UserInteraction::checkExtraFlags({UPPER_VERTEX_INDEX,
SIZE_INDEX, FILL_COLOR_INDEX, INPUT_INDEX, OUTPUT_INDEX});

upper_vertex=UserInteraction::parseCoordValue(UPPER_VERTEX_INDEX);
    size=UserInteraction::parseIntValue(SIZE_INDEX);
    fill_color=UserInteraction::parseColorValue(FILL_COLOR_INDEX);
    BmpImage myImage;
    try
    {
        myImage.drawRhombus(upper_vertex, size, fill_color);
    }
    catch(const std::invalid_argument& excp)
    {
        std::cerr << excp.what() << '\n';
        exit(ARG_VALUE_ERROR);
    }
}
```

Файл: BmpImage.h

```
#ifndef BMP_IMAGES_H
#define BMP_IMAGES_H

#include<string>
#include"bmpStructures.h"

class BmpImage
{
private:
    BmpFileHeader mBmfh;
    BmpInfoHeader mBmih;
    Rgb** mPixelArr;
    int mHeight;
    int mWidth;
    std::string filePath;

    int checkFileFormat() noexcept;
    void fileHeaderInit() noexcept;
```

```
        void infoHeaderInit() noexcept;
        int checkColor(Rgb color) noexcept;
        void checkZone(Point &leftUp, Point &rightDown) noexcept;
    public:
        BmpImage() noexcept;
        BmpImage(const std::string filePath);
        BmpImage(const int width, const int height, Rgb
defaultColor={0, 0, 0});
        void open(const std::string filePath);
        BmpFileHeader getFileHeader() noexcept;
        BmpInfoHeader getInfoHeader() noexcept;
        void printPath() noexcept;
        void save();
        void save(const std::string newFilePath);
        void resize(const int newWidth, const int newHeight, const Rgb
defaultColor={0, 0, 0});
        void drawCircle(int radius, Point center, int thickness, Rgb
lineColor, bool isFill=false, Rgb fillColor = {255, 0, 0});
        void drawLine(Point leftUp, Point rightDown, int thickness, Rgb
color);
        void drawRectangle(Point leftUp, Point rightDown, int
thickness, Rgb lineColor, bool isFill = false, Rgb fillColor = {255, 0,
0}, bool smooth=false);
        Rgb** copy(Point leftUp, Point rightDown, int &height, int
&width) noexcept;
        void paste(Point leftUp, Rgb** arr, int width, int height)
noexcept;//width and height - sizes of arr
        void invertImage();
        void blackAndWhite();
        void mirror();
        void circleOrnament(Rgb color={0, 255, 0});
        void rectangleOrnament(Rgb color, int thickness, int count);
        void semiCircleOrnament(Rgb color, int thickness, int count);
        void rotate(Point leftUp, Point rightDown, int angle);
        void drawRhombus(Point vertex, int size, Rgb fill_color);
        ~BmpImage();
    };

    #endif
```

Файл: BmpImage.cpp

```
#include"BmpImage.h"
#include"UserInteraction.h"
#include<iostream>
#include<cstring>
```

```
#include<cmath>
#include<cstdbool>

BmpImage::BmpImage() noexcept
{
    mPixelArr=nullptr;
    mHeight=0;
    mWidth=0;
}

int BmpImage::checkFileFormat() noexcept
{
    if(mBmfh.signature!=SIGNATURE_VALUE)
        return 1;
    if(mBmih.headerSize!=sizeof(BmpInfoHeader))
        return 1;
    if(mBmih.bitsPerPixel!=BITS_PER_PIXEL_VALUE)
        return 1;
    if(mBmih.compression!=COMPRESSION_VALUE)
        return 1;
    return 0;
}

BmpImage::BmpImage(const std::string filePath)
{
    this->open(filePath);
}

void BmpImage::fileHeaderInit() noexcept
{
    this->mBmfh.signature=SIGNATURE_VALUE;
    this-
>mBmfh.fileSize=mWidth*mHeight*3+sizeof(BmpInfoHeader)+sizeof(BmpFileHead
er);
    this->mBmfh.reserved1=0;
    this->mBmfh.reserved2=0;
    this-
>mBmfh.pixelArrOffset=sizeof(BmpInfoHeader)+sizeof(BmpFileHeader);
}

void BmpImage::infoHeaderInit() noexcept
{
    this->mBmih.headerSize=sizeof(BmpInfoHeader);
    this->mBmih.width=mWidth;
```

```
        this->mBmih.height=mHeight;
        this->mBmih.planes=1;
        this->mBmih.bitsPerPixel=BITS_PER_PIXEL_VALUE;
        this->mBmih.compression=COMPRESSION_VALUE;
        this->mBmih.imageSize=mWidth*mHeight*3;
        this->mBmih.xPixelsPerMeter=0;
        this->mBmih.yPixelsPerMeter=0;
        this->mBmih.colorsInColorTable=0;
        this->mBmih.importantColorCount=0;
    }

    BmpImage::BmpImage(const int width, const int height, Rgb
defaultColor)
    {
        if (width<=0 || height<=0)
            throw std::invalid_argument("Invalid image size!");
        if (checkColor(defaultColor))
            throw std::invalid_argument("Non-Rgb color type!\n");
        mPixelArr=new Rgb*[height];
        for(int i=0;i<height;i++)
        {
            mPixelArr[i]=new Rgb[width];
            for(int j=0;j<width;j++)
                mPixelArr[i][j]=defaultColor;
        }
        this->mHeight=height;
        this->mWidth=width;
        fileHeaderInit();
        infoHeaderInit();
    }

    void BmpImage::open(const std::string filePath)
    {
        std::FILE *f=fopen(filePath.c_str(), "rb");
        if(f==nullptr)
        {
            throw InvalidImageException("Unable to open bmp file!\n");
        }
        fread(&mBmfh, 1, sizeof(BmpFileHeader), f);
        fread(&mBmih, 1, sizeof(BmpInfoHeader), f);
        if(checkFileFormat())
        {
            fclose(f);
```

```
            throw       InvalidImageException("Inappropriate       file
format!\n");
        }
        mHeight=mBmih.height;
        mWidth=mBmih.width;
        mPixelArr = new Rgb*[mHeight];
        for(int i=0; i<mHeight; i++)
        {
            mPixelArr[i] = new Rgb[mWidth];
            fread(mPixelArr[i], 1, (mWidth*sizeof(Rgb)+3)&(-4), f);
        }
        fclose(f);
        this->filePath=filePath;
    }

    BmpFileHeader BmpImage::getFileHeader() noexcept
    {
        return mBmfh;
    }

    BmpInfoHeader BmpImage::getInfoHeader() noexcept
    {
        return mBmih;
    }

    void BmpImage::printPath() noexcept
    {
        std::cout<<"Image path:\t"<<filePath<<std::endl;
    }

    void BmpImage::save()
    {
        this->save(this->filePath);
    }

    void BmpImage::save(const std::string newFilePath)
    {
        std::FILE *f = fopen(newFilePath.c_str(), "wb");
        if(f==nullptr)
        {
            throw InvalidImageException("Unable to save bmp file!\n");
        }
        fwrite(&mBmfh, 1, sizeof(BmpFileHeader), f);
        fwrite(&mBmih, 1, sizeof(BmpInfoHeader), f);
```

```
        for(int i=0; i<mHeight; i++)
        {
            fwrite(mPixelArr[i], 1, (mWidth*sizeof(Rgb)+3)&(-4), f);
        }
        fclose(f);
    }

    void BmpImage::resize(const int newWidth, const int newHeight,
const Rgb defaultColor)
    {
        if(newWidth<=0 || newHeight<=0)
            throw std::invalid_argument("Invalid new image size!");
        if (checkColor(defaultColor))
            throw std::invalid_argument("Non-Rgb color type!\n");
        int tempWidth=0, tempHeight=0;
        Rgb**    temp=this->copy({0,    0},    {mWidth-1,    mHeight-1},
tempHeight, tempWidth);
        for(int i=0;i<mHeight;i++)
            delete[] mPixelArr[i];
        delete[] mPixelArr;
        mPixelArr = new Rgb*[newHeight];
        for(int i=0;i<newHeight;i++)
        {
            mPixelArr[i]=new Rgb[newWidth];
            for(int j=0;j<newWidth;j++)
                mPixelArr[i][j]=defaultColor;
        }
        this->mHeight=newHeight;
        this->mWidth=newWidth;
        fileHeaderInit();
        infoHeaderInit();
        this->paste({0, 0}, temp, tempWidth, tempHeight);
        for(int i=0;i<tempHeight;i++)
            delete[] temp[i];
        delete[] temp;
    }

    int BmpImage::checkColor(Rgb color) noexcept
    {
        return(color.r<0 || color.r>255 || color.b<0 || color.b>255 ||
color.g<0 || color.g>255);
    }
```

```
     void BmpImage::drawCircle(int radius, Point center, int thickness,
Rgb lineColor, bool isFill, Rgb fillColor)
     {
         center.y=mHeight-center.y-1;
         if(radius<0)
             throw std::invalid_argument("Negative radius!\n");
         if (thickness/2>radius)
             throw  std::invalid_argument("Thickness  is  greater  then
radius\n");
         if(thickness<=0)
             throw std::invalid_argument("Non-positive thickness!\n");
         if (checkColor(lineColor))
             throw std::invalid_argument("Non-Rgb lineColor type!\n");
         if (checkColor(fillColor))
             throw std::invalid_argument("Non-Rgb fillColor type!\n");
         int  leftBorder=0,  rightBorder=mWidth-1,  upBorder=mHeight-1,
downBorder=0;
         if(center.y-radius-thickness/2>=0)
             downBorder=center.y-radius-thickness/2;
         if(center.y+radius+thickness/2<mHeight)
             upBorder=center.y+radius+thickness/2;
         if(center.x-radius-thickness/2>=0)
             leftBorder=center.x-radius-thickness/2;
         if(center.x+radius+thickness/2<mWidth)
             rightBorder=center.x+radius+thickness/2;
         for(int i=downBorder; i<=upBorder;i++)
         {
             if(i>=0 && i<mHeight)
             {
                 for(int j=leftBorder; j<=rightBorder; j++)
                 {
                     if(j>=0 && j<mWidth)
                     {
                         if(floor(sqrt(pow(i-center.y,          2)+pow(j-
center.x, 2)))<=radius+thickness/2)
                         {
                             if(floor(sqrt(pow(i-center.y,      2)+pow(j-
center.x, 2)))>=radius-thickness/2)
                                 mPixelArr[i][j]=lineColor;
                             else if(isFill)
                                 mPixelArr[i][j]=fillColor;
                         }
                     }
                 }
```

```
            }
        }
    }

    void BmpImage::checkZone(Point &leftUp, Point &rightDown) noexcept
    {
        if(leftUp.x>rightDown.x)
        {
            int temp=leftUp.x;
            leftUp.x=rightDown.x;
            rightDown.x=temp;
        }
        if(leftUp.y>rightDown.y)
        {
            int temp=leftUp.y;
            leftUp.y=rightDown.y;
            rightDown.y=temp;
        }
    }

    void BmpImage::drawLine(Point leftUp, Point rightDown, int
thickness, Rgb color)
    {
        if(thickness<=0)
            throw std::invalid_argument("Non-positive thickness!\n");
        if (checkColor(color))
            throw std::invalid_argument("Non-Rgb color type!\n");
        leftUp.y=mHeight-leftUp.y-1;
        rightDown.y=mHeight-rightDown.y-1;
        int a=rightDown.y-leftUp.y;
        int b=leftUp.x-rightDown.x;
        int sign;
        if(abs(a)>abs(b))
            sign=1;
        else
            sign=-1;
        int f=0;
        int signa, signb;
        if (a < 0)
            signa = -1;
        else
            signa = 1;
        if (b < 0)
            signb = -1;
```

```
        else
            signb = 1;
        int x=leftUp.x;
        int y=leftUp.y;
        this->drawCircle(thickness/2, {x, mHeight-y-1}, 1, color, true,
color);
        if(x!= rightDown.x || y!=rightDown.y)
        {
            if(sign==-1)
            {
                do
                {
                    f+=a*signa;
                    if(f>0)
                    {
                        f-=b*signb;
                        y+=signa;
                    }
                    x-=signb;
                    this->drawCircle(thickness/2, {x, mHeight-y-1}, 1,
color, true, color);

                }
                while(x!= rightDown.x || y!=rightDown.y);
            }
            else
            {
                do
                {
                    f+=b*signb;
                    if(f>0)
                    {
                        f-=a*signa;
                        x-=signb;
                    }
                    y+=signa;
                    this->drawCircle(thickness/2, {x, mHeight-y-1}, 1,
color, true, color);
                }
                while(x!=rightDown.x || y!=rightDown.y);
            }
        }
    }
```

```
    void BmpImage::drawRectangle(Point leftUp, Point rightDown, int
thickness, Rgb lineColor, bool isFill, Rgb fillColor, bool smooth)
    {
        checkZone(leftUp, rightDown);
        if(thickness<=0)
            throw std::invalid_argument("Non-positive thickness!\n");
        if (checkColor(lineColor))
            throw std::invalid_argument("Non-Rgb lineColor type!\n");
        if (checkColor(fillColor))
            throw std::invalid_argument("Non-Rgb fillColor type!\n");
        leftUp.y=mHeight-leftUp.y-1;
        rightDown.y=mHeight-rightDown.y-1;
        if(smooth==false)
        {
            for(int i=rightDown.y;i<=leftUp.y;i++)
                if(i>=0 && i<mHeight)
                {
                    for(int j=leftUp.x;j<=rightDown.x;j++)
                        if(j>=0 && j<mWidth)
                        {
                            if(i<rightDown.y+thickness  ||  i>leftUp.y-
thickness || j<leftUp.x+thickness || j>rightDown.x-thickness)
                                mPixelArr[i][j]=lineColor;
                            else if(isFill)
                                mPixelArr[i][j]=fillColor;
                        }
                }
        }
        else
        {
            if(isFill)
            {
                for(int i=rightDown.y;i<=leftUp.y;i++)
                    if(i>=0 && i<mHeight)
                    {
                        for(int j=leftUp.x;j<=rightDown.x;j++)
                            if(j>=0 && j<mWidth)
                                mPixelArr[i][j]=fillColor;
                    }
            }
            leftUp.y=mHeight-leftUp.y-1;
            rightDown.y=mHeight-rightDown.y-1;
            this->drawLine({leftUp.x,      leftUp.y},      {rightDown.x,
leftUp.y}, thickness, lineColor);
```

```
            this->drawLine({leftUp.x,       leftUp.y},       {leftUp.x,
rightDown.y}, thickness, lineColor);
            this->drawLine({rightDown.x,    leftUp.y},    {rightDown.x,
rightDown.y}, thickness, lineColor);
            this->drawLine({leftUp.x,    rightDown.y},    {rightDown.x,
rightDown.y}, thickness, lineColor);
        }
    }

    Rgb** BmpImage::copy(Point leftUp, Point rightDown, int &height,
int &width) noexcept
    {
        checkZone(leftUp, rightDown);
        if(leftUp.y<0)
            leftUp.y=0;
        if(leftUp.x<0)
            leftUp.x=0;
        if(rightDown.x>=mWidth)
            rightDown.x=mWidth-1;
        if(rightDown.y>=mHeight)
            rightDown.y=mHeight-1;
        if(leftUp.y>=mHeight || leftUp.x>=mWidth || rightDown.y<0 ||
rightDown.x<0)
        {
            height=0;
            width=0;
            return nullptr;
        }
        leftUp.y=mHeight-leftUp.y-1;
        rightDown.y=mHeight-rightDown.y-1;
        height=leftUp.y-rightDown.y+1;
        width=rightDown.x-leftUp.x+1;
        Rgb** copiedArray = new Rgb*[height];
        for(int i=rightDown.y;i<=leftUp.y;i++)
        {
            copiedArray[i-rightDown.y]= new Rgb[width];
            for(int j=leftUp.x;j<=rightDown.x;j++)
            {
                copiedArray[i-rightDown.y][j-leftUp.x]=mPixelArr[i][j];
            }
        }
        return copiedArray;
    }
```

```
    void BmpImage::paste(Point leftUp, Rgb** arr, int width, int
height) noexcept//width and height - sizes of arr
    {
        leftUp.y=mHeight-leftUp.y-1;
        for(int i=0; i<height;i++)
        {
            for(int j=0; j<width;j++)
            {
                if(leftUp.y-height+i+1>=0          &&          leftUp.y-
height+i+1<mHeight && leftUp.x+j<mWidth && leftUp.x+j>=0)
                {
                    mPixelArr[leftUp.y-
height+1+i][j+leftUp.x]=arr[i][j];
                }
            }
        }
    }

    void invertPixel(Rgb &pixel)
    {
        pixel.b=255-pixel.b;
        pixel.g=255-pixel.g;
        pixel.r=255-pixel.r;
    }

    void BmpImage::invertImage()
    {
        for(int i=0;i<mHeight;i++)
            for(int j=0;j<mWidth;j++)
                invertPixel(mPixelArr[i][j]);
    }

    void makeGreyPixel(Rgb& pixel)
    {
        int  grey  =  static_cast<int>(0.299*static_cast<double>(pixel.r)
+                  0.587*static_cast<double>(pixel.r)                  +
0.114*static_cast<double>(pixel.b));
        pixel.b=grey;
        pixel.r=grey;
        pixel.g=grey;
    }

    void BmpImage::blackAndWhite()
    {
```

```
        for(int i=0;i<mHeight;i++)
            for(int j=0;j<mWidth;j++)
                makeGreyPixel(mPixelArr[i][j]);
    }

    void BmpImage::mirror()
    {
        for(int i=0; i<mHeight;i++)
        {
            for(int j=0;j<(mWidth/2);j++)
            {
                Rgb temp=mPixelArr[i][j];
                mPixelArr[i][j]=mPixelArr[i][mWidth-j-1];
                mPixelArr[i][mWidth-j-1]=temp;
            }
        }
    }
    void BmpImage::circleOrnament(Rgb color)
    {
        if (checkColor(color))
            throw std::invalid_argument("Non-Rgb color type!\n");
        int radius;
        if (mHeight>mWidth)
            radius=mWidth/2;
        else
            radius=mHeight/2;
        for(int i=0;i<mHeight;i++)
        {
            for(int j=0;j<mWidth;j++)
            {
                if(floor(sqrt(pow(i-(mHeight/2),    2)+pow(j-(mWidth/2),
2)))>radius)
                    mPixelArr[i][j]=color;
            }
        }
    }

    void  BmpImage::rectangleOrnament(Rgb  color,  int  thickness,  int
count)
    {
        if (checkColor(color))
            throw std::invalid_argument("Non-Rgb color type!\n");
        if(thickness<=0)
            throw std::invalid_argument("Non-positive thickness!\n");
```

```
        if(count<=0)
            throw std::invalid_argument("Non-positive count!\n");
        if(thickness> mWidth/2 || thickness> mHeight/2)
            throw std::invalid_argument("Too big thickness!\n");
        if(2*(thickness*2*count-1)>mWidth    ||    2*(thickness*2*count-
1)>mHeight)
            throw std::invalid_argument("Too big count!\n");
        for(int i=0; i<count; i++)
        {
            this->drawRectangle({2*i*thickness,          2*i*thickness},
{mWidth-2*i*thickness-1, mHeight-2*i*thickness-1}, thickness, color);
        }
    }

    void  BmpImage::semiCircleOrnament(Rgb  color,  int  thickness,  int
count)
    {
        if (checkColor(color))
            throw std::invalid_argument("Non-Rgb color type!\n");
        if(thickness<=0)
            throw std::invalid_argument("Non-positive thickness!\n");
        if(count<=0)
            throw std::invalid_argument("Non-positive count!\n");
        int
radiusWidth=ceil(static_cast<double>(mWidth)/(2.0*static_cast<double>(cou
nt)));
        int
radiusHeight=ceil(static_cast<double>(mHeight)/(2.0*static_cast<double>(c
ount)));
        if(radiusHeight<1 || radiusWidth<1)
            throw std::invalid_argument("Too big count!\n");
        if(radiusHeight<thickness || radiusWidth<thickness)
            throw std::invalid_argument("Too big thickness\n");
        for(int i=0;i<count;i++)
        {
            this->drawCircle(radiusHeight,  {0,  (2*i+1)*radiusHeight},
thickness, color);
            this->drawCircle(radiusHeight,             {this->mWidth-1,
(2*i+1)*radiusHeight}, thickness, color);
            this->drawCircle(radiusWidth,   {(2*i+1)*radiusWidth,   0},
thickness, color);
            this->drawCircle(radiusWidth,  {(2*i+1)*radiusWidth,  this-
>mHeight-1}, thickness, color);
        }
```

```
    }

    void BmpImage::drawRhombus(Point vertex, int size, Rgb fill_color)
    {
        if (checkColor(fill_color))
            throw std::invalid_argument("Non-Rgb color type!\n");
        if(size<=0)
            throw std::invalid_argument("Non-positive size!\n");
        int a=floor(sqrt(pow(size, 2)/2.0));
        int j=0;
        for(int i=0; i<2*a;i++)
        {
            drawLine({vertex.x-j, vertex.y+i},{vertex.x+j, vertex.y+i},
1, fill_color);
            if(i>a)
                j--;
            else
                j++;
        }
    }

    Point getRotationAxis(Point leftUp, Point rightDown, int width, int
height)
    {
        Point      axis={(rightDown.x+leftUp.x)/2      -      height/2,
(rightDown.y+leftUp.y)/2 - width/2};
        return axis;
    }

    void BmpImage::rotate(Point leftUp, Point rightDown, int angle)
    {
        int height=0, width=0;
        if(angle!=90 && angle!=180 && angle!=270)
            throw std::invalid_argument("Invalid rotation angle!");
        Rgb**    pixelArray    =    this->copy(leftUp,    {rightDown.x-1,
rightDown.y-1}, height, width);
        if(angle==180)
        {
            Rgb** newPixelArray = new Rgb*[height];
            for(int i=0;i<height; i++)
                newPixelArray[i]=new Rgb[width];
            for(int i=0;i<height;i++)
                for(int j=0;j<width;j++)
```

```
                    newPixelArray[i][j]=pixelArray[height-i-1][width-j-
1];
            this->paste(leftUp, newPixelArray, width, height);
        }
        else
        {
            Rgb** newPixelArray = new Rgb*[width];
            for(int i=0;i<width; i++)
                newPixelArray[i]=new Rgb[height];
            if(angle==90)
            {
                for(int i=0;i<height;i++)
                    for(int j=0;j<width;j++)
                        newPixelArray[j][i]=pixelArray[height-i-1][j];
            }
            else
            {
                for(int i=0;i<height;i++)
                    for(int j=0;j<width;j++)
                        newPixelArray[j][i]=pixelArray[i][width-j-1];
            }
            this->paste(getRotationAxis(leftUp,    rightDown,    width,
height), newPixelArray, height, width);
        }
    }

    BmpImage::~BmpImage()
    {
        for(int i=0;i<mHeight;i++)
            delete[] mPixelArr[i];
        delete[] mPixelArr;
}
```