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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка bmp-файлов

Студент гр. 3344 _____ Сербиновский Ю.М.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2024

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Сербиновский Ю.М.

Группа 3344

Тема работы: обработка bmp-файлов

Исходные данные:

- 24 бита на цвет
- без сжатия
- файл может не соответствовать формату BMP, т.е. необходимо проверка на BMP формат (дополнительно стоит помнить, что версий у формата несколько). Если файл не соответствует формату BMP или его версии, то программа должна завершиться с соответствующей ошибкой.
- обратите внимание на выравнивание; мусорные данные, если их необходимо дописать в файл для выравнивания, должны быть нулями.
- обратите внимание на порядок записи пикселей
- все поля стандартных BMP заголовков в выходном файле должны иметь те же значения что и во входном (разумеется, кроме тех, которые должны быть изменены).

Содержание пояснительной записки:

1. Содержание
2. Введение
3. Задание варианта
4. Функции программы
5. Структуры и классы программы
6. Файловая структура программы
7. Описание сборки проекта
8. Примеры работы программы

9. Примеры обработки ошибок
10. Заключение
11. Код программы

Предполагаемый объем пояснительной записки:
Не менее 45 страниц.

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 15.05.2024

Дата защиты реферата: 15.05.2024

Студент _____ Сербиновский Ю.М.

Преподаватель _____ Глазунов С.А.

АННОТАЦИЯ

Программа представляет собой набор инструментов для работы с bmp-файлами, взаимодействовать с которыми можно через командную строку. В программе реализованы такие возможности, как рисование окружности, прямоугольника и узоров по определенным параметрам, а также поворот изображения на 90, 180 или 270 градусов. В процессе работы программы обрабатываются исключительные случаи связанные как с работой CLI проекта, так и с непосредственной обработкой входного изображения. Результатом работы программы является обработанный bmp-файл.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью работы является создание программы для работы с bmp-файлами. Для достижения поставленной цели необходимо решить такие задачи как: реализация надежного CLI для удобного считывания входных данных, функционала для импорта и экспорта байтовой информации входного и выходного изображения, методов для редактирования изображений, а также обработка исключительных случаев, написание Makefile для сборки программы.

1. ОПИСАНИЕ ВАРИАНТА РАБОТЫ

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).
    - Цветом. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту, например: `--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`
4. Рисование окружности. Флаг для выполнения данной операции: `--circle`. Окружность определяется:
   - 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
   - толщиной линии окружности. Флаг `--thickness`. На вход принимает число больше 0
   - цветом линии окружности. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)
   - окружность может быть залитой или нет. Флаг `--fill`. Работает как бинарное значение: флага нет – false , флаг есть – true.
   - цветом которым залита сама окружность, если пользователем выбрана залитая окружность. Флаг `--fill_color` (работает аналогично флагу `--color`)

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

2. ОПИСАНИЕ ПРОГРАММЫ

2.1. Описание функций программы

- int main() - главная функция программы которая вызывает другие функции проекта.
- std::unordered_map<std::string, std::string> getFlags(int argc, char** argv) - считывает входные данные из консоли, проводит первичную проверку входных данных и записывает их в хэш-таблицу.
- std::string run_this(std::unordered_map<std::string, std::string> & flags_table) – в зависимости от введенных аргументов возвращает название функции, которую необходимо выполнить
- int value_check (std::string key, std::unordered_map<std::string, std::string> & flags_table) – проверяет значения введенных аргументов на соответствие шаблонам, описанным в задании.
- int validate_dependecies (std::string function, std::unordered_map <std::string, std::string> & flags_table) – проверяет, является ли набор введеных аргументов командной строки достаточным или избыточным для выполнения того или иного функционала.
- Rectangle rect_struct (std::unordered_map<std::string, std::string> & flags_table) - проверяет значения параметров для рисования прямоугольника и записывает их в поля соответствующей структуры.
- circ_struct – проверяет значения параметров для рисования окружности и записывает их в поля соответствующей структуры.
- orn_struct – проверяет значения параметров для рисования узоров и записывает их в поля соответствующей структуры.
- rot_struct – проверяет значения параметров для поворота изображения и записывает их в поля соответствующей структуры.
- Image::readBMP – считывает байтовую информацию об изображении и записывает в поля класса Image.
- Image::exportBMP – формирует конечное изображение.

- Image::line – рисует линию, используя алгоритм Брезенхэма. Метод используется для удобного рисования прямоугольника.
- Image::rectangle – рисует прямоугольник по заданным координатам.
- Image::circle – рисует окружность с использованием алгоритма Брезенхэма.
- Image::rotate – поворачивает прямоугольную область изображения на 90, 180 или 270 градусов вокруг центра.
- Image::ornament – доступно три узора: прямоугольники, окружность или полуокружности. Метод рисует один из трех узоров на выбор.

2.2. Структуры и классы программы

- FileHeader – структура, представляющая собой файловый заголовок bmp-файла.
- InfoHeader – структура, представляющая собой информационный заголовок bmp-файла.
- Color – структура, хранит информацию о цвете одного пикселя в формате RGB.
- Rectangle – структура, хранит информацию о параметрах функции Image::rectangle.
- Circle – структура, хранит информацию о параметрах функции Image::circle
- Ornament – структура, хранит информацию о параметрах функции Image::ornament.
- Rotate – структура, хранит информацию о параметрах функции Image::rotate.
- Image – представляет собой набор всех реализованных в программе методов для обработки, импорта и экспорта изображений. Приватные поля класса хранят всю необходимую информацию об изображении.

2.3. Описание файловой структуры программы

- main.cpp – содержит функцию int main() проекта.
- checking_flags.cpp – файл, в котором реализовано считывание данных из консоли.
- checking_flags.h – заголовочный файл, содержащий прототип функции для считывания данных.
- find_function_to_run.cpp – файл, определяющий по входным данным, какой функционал функции должен быть использован.
- find_function_to_run.h - заголовочный файл, содержащий прототип функции из find_function_to_run.cpp.
- validations.cpp - файл, в котором реализована проверка введенных в консоль данных.
- validations.h - заголовочный файл, содержащий прототипы функций из validations.cpp.
- messages.h – файл, в котором описаны различные сообщения для пользователя, включая справки и сообщения об ошибках.
- struct_work.cpp – файл, в котором реализована запись данных, полученных из консоли в соответствующие структуры.
- struct_work.h - заголовочный файл, содержащий прототипы функций из struct_work.cpp.
- structs.h – файл, в котором собраны все структуры используемые в тех или иных местах проекта.
- image_work.cpp – файл, в котором реализованы все методы класса Image, отвечающие за обработку изображений.
- io.cpp – файл, в котором реализованы все методы класса Image, отвечающие за импорт и экспорт изображений.
- one_class.h – файл, содержащий единственный класс Image проекта с прототипами методов для работы с изображениями.

2.4. Описание сборки проекта

Makefile отвечает за сборку проекта:

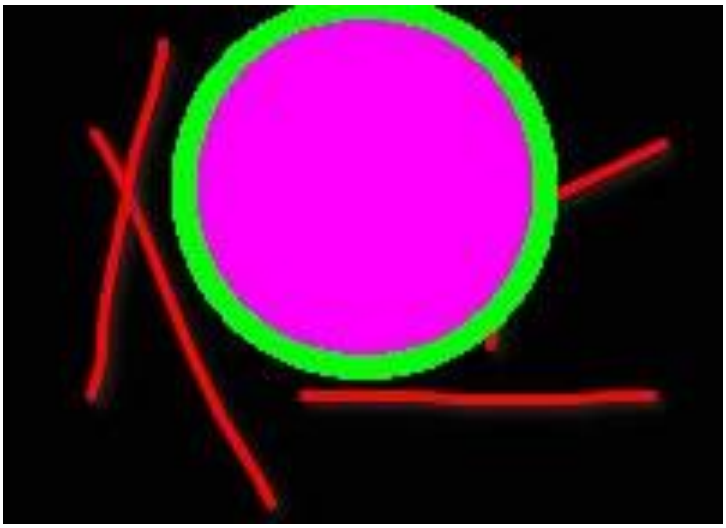
- CC = gcc: Указывает компилятор, используемый для сборки.
- CPPFLAGS = -lstdc++ -lm -O3: Определяет флаги для компиляции и линковки, включая использование стандартной библиотеки C++ и библиотеки math.
- MAKEFLAGS += -j4: Задает количество параллельных задач для ускорения сборки.
- SRCDIRS = . input image_methods other: Перечисляет каталоги, содержащие исходные файлы.
- SOURCES = $(foreach dir, $(SRCDIRS), $(wildcard $(dir)/*.cpp)): Собирает список всех исходных файлов .cpp из указанных каталогов.
- OBJECTS = $(patsubst %.cpp, %.o, $(SOURCES)): Создает список объектных файлов, соответствующих исходным файлам.
- %.o: %.cpp: Правило для компиляции исходных файлов в объектные файлы.
- cw: Основная цель, которая собирает исполняемый файл cw из объектных файлов
- clean: Вспомогательная цель для удаления всех объектных файлов и исполняемого файла cw.

3. ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

- **–circle:**

  Ввод: ./cw --circle --center 100.50 --radius 50 --color 0.255.0 --fill true --fill_color 255.0.255 --thickness 5 -o output.bmp input.bmp

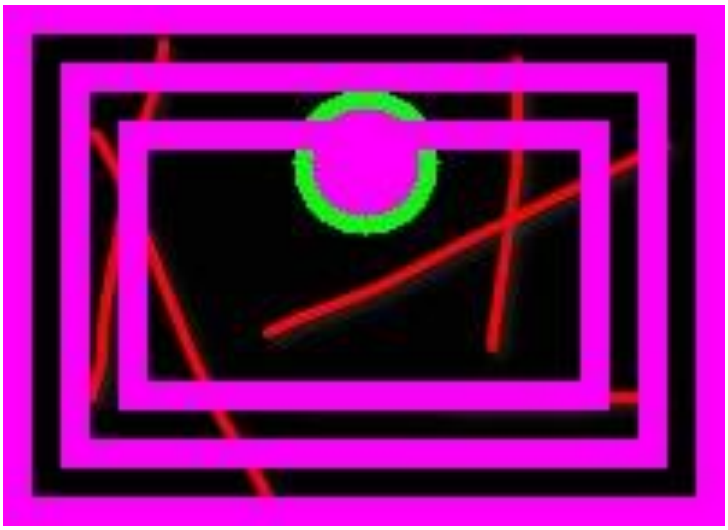
  Вывод:



- **–ornament:**

  Ввод: ./cw --ornament --thickness 8 --color 255.0.255 --count 3 --pattern rectangle -o output.bmp input.bmp

  Вывод:



- **–rotate:**

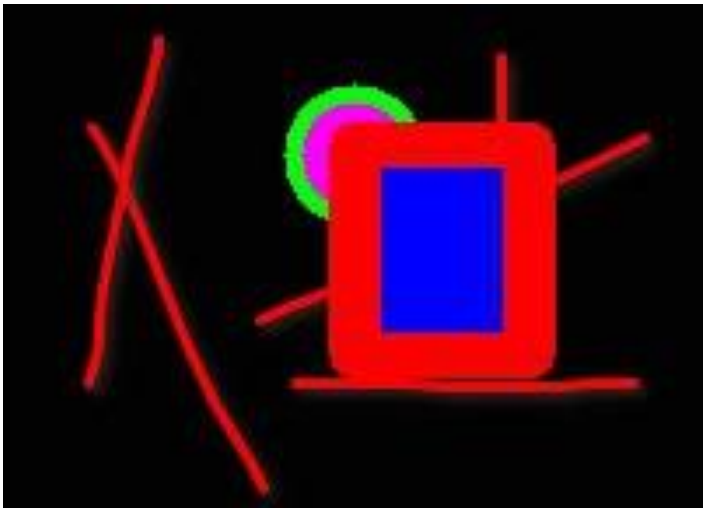
  Ввод: ./cw --rotate --left_up 0.100 --right_down 100.50 --angle 180 -o output.bmp input.bmp

  Вывод:

- **–rect:**

  Ввод: ./cw --rect --left_up 100.40 --right_down 150.100 --color 255.0.0 --thickness 10 --fill true --fill_color 0.0.255 -o output.bmp input.bmp

  Вывод:



4. ПРИМЕРЫ ОБРАБОТКИ ОШИБОК

- **Неправильные аргументы:**

  Ввод: ./cw --rect --right_down 150.100 --color 255.0.0 --thickness 10 --fill true --fill_color 52.pisyat.dva -o kitty.bmp sample.bmp

  Вывод:

```
Something wrong with arguments. Maybe there are too many arguments or not enough of them.
Also the problem may be in incorrect data passed to the arguments.
Bad arguments for rect
Use --help or -h for more information.
```

- **Несуществующий входной файл:**

  Ввод: ./cw --rect --left_up 100.40 --right_down 150.100 --color 255.0.0 --thickness 10 --fill true --fill_color 0.0.255 -o kitty.bmp pisyat-dva.bmp

  Вывод:

```
Failed to open the input file
```

ЗАКЛЮЧЕНИЕ

Была написана программа на языке программирования C++, позволяющая редактировать изображения формата bmp. Все поставленные задачи были выполнены, а цель достигнута. Использование getopt позволило создать удобный и гибкий CLI, предоставляющий пользователю доступ к функциям программы посредством набора команд. Makefile упростил процесс сборки и настройки проекта, автоматизировав выполнение рутинных операций. В случае некорректных входных данных пользователю выводятся сообщения об ошибках. Таким образом, разработанная программа демонстрирует практическое применение изученных технологий и библиотек для решения задачи редактирования графических файлов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Базовые сведения к выполнению курсовой и лабораторных работ по дисциплине «программирование». Второй семестр: учеб.-метод. пособие др. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2024. 36 с.

2. Geeksforgeeks. URL: https://www.geeksforgeeks.org (Дата обращения 10.05.2024)

3. Cplusplus. URL: https://cplusplus.com/reference/ (Дата обращения 12.05.2024)

ПРИЛОЖЕНИЕ А.
КОД ПРОГРАММЫ

Makefile

```
CC = gcc
CPPFLAGS = -lstdc++ -lm -O3
MAKEFLAGS += -j4
SRCDIRS = . input image_methods other
SOURCES = $(foreach dir, $(SRCDIRS), $(wildcard $(dir)/*.cpp))
OBJECTS = $(patsubst %.cpp, %.o, $(SOURCES))

%.o: %.cpp
	$(CC) -c $< -o $@

cw: $(OBJECTS)
	$(CC) $(OBJECTS) $(CPPFLAGS) -o cw

clean:
	rm -f $(OBJECTS) cw
```

main.cpp

```
#include "input/checking_flags.h"
#include "input/find_function_to_run.h"
#include "image_methods/one_class.h"
#include "other/struct_work.h"
#include "other/structs.h"
#include "other/messages.h"

int main(int argc, char** argv)
{
    if (argc < 2)
    {
        std::cout << "It seems, you forgot to pick the option" <<
std::endl;
        std::cout << HELP << std::endl;
        exit(40);
    }


    std::unordered_map<std::string, std::string> flags_table =
getFlags(argc, argv);

    if (flags_table.find("help") != flags_table.end())
    {
        std::cout << "Course work for option 5.1, created by Yurii
Serbinovskii." << std::endl;
        std::cout << PRINT_HELP << std::endl;
        exit(0);
    }
    std::string function_to_run = run_this(flags_table);
```

```
    Image img;
    img.readBMP(flags_table["input"], function_to_run);

    if (flags_table["output"] == flags_table["input"])
    {
        std::cerr << OUTPUT << std::endl;
        exit(43);
    }

    if (function_to_run == "rect")
    {
        Rectangle rect = rect_struct(flags_table);
        img.rectangle(rect);
    }
    else if (function_to_run == "circle")
    {
        Circle circ = circ_struct(flags_table);
        img.circle(circ);
    }
    else if (function_to_run == "ornament")
    {
        Ornament orn = orn_struct(flags_table);
        img.ornament(orn);
    }
    else if (function_to_run == "rotate")
    {
        Rotate rot = rot_struct(flags_table);
        img.rotate(rot);
    }
    img.exportBMP(flags_table["output"]);
}
```

checking_flags.cpp

```
#include "checking_flags.h"

struct option long_opt[]={
    {"input", 1, 0, 'i'},
    {"output", 1, 0, 'o'},
    {"info", 0, 0, 0},
    {"help", 0, 0, 'h'},
    {"rect", 0, 0, 0},
    {"left_up", 1, 0, 0},
    {"right_down", 1, 0, 0},
    {"thickness", 1, 0, 0},
    {"color", 1, 0, 0},
    {"fill_color", 1, 0, 0},
    {"ornament", 0, 0, 0},
    {"pattern", 1, 0, 0},
```

```
    {"color", 1, 0, 0},
    {"count", 1, 0, 0},
    {"rotate", 0, 0, 0},
    {"angle", 1, 0, 0},
    {"circle", 0, 0, 0},
    {"center", 1, 0, 0},
    {"radius", 1, 0, 0},
    {"fill", 2, 0, 0},
    {0, 0, 0, 0}
};


std::unordered_map<std::string, std::string> getFlags(int argc, char**
argv)
{
    int optindx;
    int opt;
    opterr = 0;
    std::string flag;
    std::unordered_map<std::string, std::string> flags_table;

    while((opt = getopt_long(argc, argv, "i:o:h", long_opt, &optindx)) !=
-1) {
        switch (opt)
        {
        case '?':
            std::cerr << UNKNOWN_ARG << argv[optind - 1] << std::endl;
            std::cout << HELP << std::endl;
            exit(44);
            break;
        case 'i':
            flags_table["input"] = optarg;
            break;
        case 'o':
            flags_table["output"] = optarg;
            break;
        case 'h':
            flags_table["help"] = "";
            break;
        default:
            if (flags_table.find("input") == flags_table.end()) {
                if (optind < argc)
                {
                    flags_table["input"] = argv[argc - 1];
                }
                else
                {
                    std::cerr << INPUT << std::endl;
                    exit(43);
                }
```

```
            }
            if (optarg) {
                flags_table[long_opt[optindx].name] = optarg;
            }
            else {
                flags_table[long_opt[optindx].name] = "";
            }
            break;
        }
    }
    return flags_table;
}
```

checking_flags.h

```
#pragma once

#include <iostream>
#include <getopt.h>
#include <unordered_map>
#include <string>
#include "../other/messages.h"

std::unordered_map<std::string, std::string> getFlags(int argc, char**
argv);
```

find_function_to_run.cpp

```
#include "find_function_to_run.h"
#include "validations.h"


std::string run_this(std::unordered_map<std::string, std::string> &
flags_table) {
    if (flags_table.find("info") != flags_table.end()) {
        if (validate_dependecies("info", flags_table)) {
            return "info";
        }
        else {
            std::cerr << DEPENDECIES << "info" << std::endl;
            std::cout << HELP << std::endl;
            exit(41);
        }
    }
    else if (flags_table.find("rect") != flags_table.end()) {
        if (validate_dependecies("rect", flags_table)) {
            return "rect";
        }
        else {
            std::cerr << DEPENDECIES << "rect" << std::endl;
            std::cout << HELP << std::endl;
            exit(41);
```

```
        }
    }
    else if (flags_table.find("ornament") != flags_table.end()) {
        if (validate_dependecies("ornament", flags_table)) {
            return "ornament";
        }
        else {
            std::cerr << DEPENDECIES << "ornament" << std::endl;
            std::cout << HELP << std::endl;
            exit(41);
        }
    }
    else if (flags_table.find("rotate") != flags_table.end()) {
        if (validate_dependecies("rotate", flags_table)) {
            return "rotate";
        }
        else {
            std::cerr << DEPENDECIES << "rotate" << std::endl;
            std::cout << HELP << std::endl;
            exit(41);
        }
    }
    else if (flags_table.find("circle") != flags_table.end()) {
        if (validate_dependecies("circle", flags_table)) {
            return "circle";
        }
        else {
            std::cerr << DEPENDECIES << "circle" << std::endl;
            std::cout << HELP << std::endl;
            exit(41);
        }
    }
    else {
        std::cerr << "It seems, you forgot to enter the functions" <<
std::endl;
        std::cout << HELP << std::endl;
        exit(40);
    }
}
```

find_function_to_run.h

```
#pragma once

#include <iostream>
#include <string>
#include <unordered_map>
```

```
std::string   run_this(std::unordered_map<std::string,   std::string>   &
flags_table);
```

validations.cpp

```
#include "validations.h"


int    value_check(std::string    key,    std::unordered_map<std::string,
std::string> & flags_table)
{
    std::regex cords_reg ("^[0-9]+\\.[0-9]+$");
    std::regex   color_reg   ("^(([0-1]?[0-9]?[0-9])|(2[0-4][0-9])|(25[0-
5]))\\.(([0-1]?[0-9]?[0-9])|(2[0-4][0-9])|(25[0-5]))\\.(([0-1]?[0-9]?[0-
9])|(2[0-4][0-9])|(25[0-5]))$");
    std::regex positive_reg ("^[0-9]+$");


    if (key == "color" || key == "fill_color")
    {
        std::smatch color_match;
        if (std::regex_search(flags_table[key], color_match, color_reg))
        {
            return 1;
        }
        return 0;
    }
    if (key == "left_up" || key == "right_down" || key == "center")
    {
        std::smatch cords_match;
        if (std::regex_search(flags_table[key], cords_match, cords_reg))
        {
            return 1;
        }
        return 0;
    }
    if (key == "thickness" || key == "count" || key == "radius")
    {
        std::smatch positive_match;
        if      (std::regex_search(flags_table[key],      positive_match,
positive_reg))
        {
            return 1;
        }
        return 0;
    }
    if (key == "angle")
    {
        if  (flags_table[key]  ==  "90"  ||  flags_table[key]  ==  "180"  ||
flags_table[key] == "270")
        {
```

```
            return 1;
        }
        return 0;
    }
    if (key == "pattern")
    {
        if (flags_table[key] == "circle" || flags_table[key] ==
"semicircles" || flags_table[key] == "rectangle")
        {
            return 1;
        }
        return 0;
    }
    return 1;
}


int validate_dependecies(std::string function,
std::unordered_map<std::string, std::string> & flags_table) {
    std::vector<std::string> some_keys;
    if (function == "info")
    {
        some_keys = {"info", "input"};
    }
    if (function == "rect") {
        some_keys = {"rect", "left_up", "right_down", "thickness",
"color", "fill", "fill_color", "input", "output"};
    }
    if (function == "ornament") {
        some_keys = {"ornament", "pattern", "color", "thickness",
"count", "input", "output"};
    }
    if (function == "rotate") {
        some_keys = {"rotate", "left_up", "right_down", "angle", "input",
"output"};
    }
    if (function == "circle") {
        some_keys = {"circle", "center", "radius", "thickness", "color",
"fill", "fill_color", "input", "output"};
    }

    for (int i = 0; i < some_keys.size(); i++)
    {
        if (flags_table.find(some_keys[i]) != flags_table.end())
        {
            if (!value_check(some_keys[i], flags_table))
            {
                return 0;
            }
        }
```

```
        else if (some_keys[i] == "fill_color" || some_keys[i] == "fill")
        {
            continue;
        }
        else
        {
            return 0;
        }
    }

    return 1;
}
```

validations.h

```
#pragma once

#include <iostream>
#include <string>
#include <regex>
#include "../other/messages.h"
#include <unordered_map>

int value_check(std::string function, std::unordered_map<std::string,
std::string> & flags_table);
int validate_dependecies(std::string function,
std::unordered_map<std::string, std::string> & flags_table) ;
```

messages.h

```
#pragma once


#define PRINT_HELP "You should pick one of six options: \n\
1) --help - shows help info, if you need some\n\
2) --info - shows information about input file\n\
3) --rect - draws rectangle. It has five important and one optioanal
arguments: --left_up & --right_down (format: \"x.y\"), --thickness
(positive number),\
 --color (format: \"r.g.b\"), --fill (true or false), --fill_color
(important, if --fill = true; format: \"r.g.b\")\n\
4) --ornament - draws beatiful ornaments. Arguments: --pattern (circle,
or rectangle, or semicircle), left_up & right_down (format: \"x.y\")\n\
5) --circle - draws circle. Arguments: --center (format: \"x.y\"), --
radius (positive number), --thickness (positive number),\
 --color (format: \"r.g.b\"), --fill (true or false), --fill_color
(important, if --fill = true; format: \"r.g.b\")\n\
6) --rotate - rotates the rectangle area by an angle. Arguments: --
left_up & --right_down (format: \"x.y\"), --angle (90, or 180, or 270)\n\
Also, important arguments: -i (--input) & -o (--output) - for input and
output file.\
```

```
 If --input is missing, last argument of comand line would be consider as
input file."


#define HELP "Use --help or -h for more information."
#define UNKNOWN_ARG "Unknown argument or missing value for "
#define DEPENDECIES "Something wrong with arguments. Maybe there are too
many arguments or not enough of them. \n\
Also the problem may be in incorrect data passed to the arguments.\n\
Bad arguments for "
#define OUTPUT "Output file have the same name as input file, or output
file is missing"
#define INPUT "Input file is missing"
#define WRONG_FILE "That's the wrong file"
```

struct_work.cpp

```
#include "struct_work.h"


Rectangle   rect_struct(std::unordered_map<std::string,   std::string>   &
flags_table)
{
    Rectangle rect;
    rect.left_up[0]     =     std::stoi(flags_table["left_up"].substr(0,
flags_table["left_up"].find('.')));
    rect.left_up[1]                                                    =
std::stoi(flags_table["left_up"].substr(flags_table["left_up"].find('.')
+ 1, std::string::npos - flags_table["left_up"].find('.')));



    rect.right_down[0]   =   std::stoi(flags_table["right_down"].substr(0,
flags_table["right_down"].find('.')));
    rect.right_down[1]                                                 =
std::stoi(flags_table["right_down"].substr(flags_table["right_down"].find
('.') + 1, std::string::npos - flags_table["right_down"].find('.')));

    rect.thickness = std::stoi(flags_table["thickness"]);
    rect.color.b        =        std::stoi(flags_table["color"].substr(0,
flags_table["color"].find('.')));
    rect.color.g                                                       =
std::stoi(flags_table["color"].substr(flags_table["color"].find('.') + 1,
flags_table["color"].rfind('.') - flags_table["color"].find('.')));
    rect.color.r                                                       =
std::stoi(flags_table["color"].substr(flags_table["color"].rfind('.')   +
1, std::string::npos - flags_table["color"].rfind('.')));

    if     (flags_table.find("fill")     !=     flags_table.end()     &&
flags_table.find("fill_color") != flags_table.end())
    {
```

```
        rect.fill = 1;
        rect.fill_color.b = std::stoi(flags_table["fill_color"].substr(0,
flags_table["fill_color"].find('.')));
        rect.fill_color.g                                                =
std::stoi(flags_table["fill_color"].substr(flags_table["fill_color"].find
('.')       +       1,       flags_table["fill_color"].rfind('.')       -
flags_table["fill_color"].find('.')));
        rect.fill_color.r                                                =
std::stoi(flags_table["fill_color"].substr(flags_table["fill_color"].rfin
d('.') + 1, std::string::npos - flags_table["fill_color"].rfind('.')));
    }
    else if (flags_table.find("fill") != flags_table.end())
    {
        std::cerr << "Where is fill_color?" << std::endl;
        exit(46);
    }
    else
    {
        rect.fill = 0;
    }

    return rect;
}

Circle    circ_struct(std::unordered_map<std::string,    std::string>    &
flags_table)
{
    Circle circ;
    circ.center[0]      =      std::stoi(flags_table["center"].substr(0,
flags_table["center"].find('.')));
    circ.center[1]                                                       =
std::stoi(flags_table["center"].substr(flags_table["center"].find('.')  +
1, std::string::npos - flags_table["center"].find('.')));

    circ.radius = std::stoi(flags_table["radius"]);

    circ.thickness = std::stoi(flags_table["thickness"]);
    circ.color.b        =        std::stoi(flags_table["color"].substr(0,
flags_table["color"].find('.')));
    circ.color.g                                                         =
std::stoi(flags_table["color"].substr(flags_table["color"].find('.') + 1,
flags_table["color"].rfind('.') - flags_table["color"].find('.')));
    circ.color.r                                                         =
std::stoi(flags_table["color"].substr(flags_table["color"].rfind('.')   +
1, std::string::npos - flags_table["color"].rfind('.')));



    if     (flags_table.find("fill")     !=     flags_table.end()     &&
flags_table.find("fill_color") != flags_table.end())
```

```
    {
        circ.fill = 1;
        circ.fill_color.b = std::stoi(flags_table["fill_color"].substr(0,
flags_table["fill_color"].find('.')));
        circ.fill_color.g                                                =
std::stoi(flags_table["fill_color"].substr(flags_table["fill_color"].find
('.')       +       1,       flags_table["fill_color"].rfind('.')       -
flags_table["fill_color"].find('.')));
        circ.fill_color.r                                                =
std::stoi(flags_table["fill_color"].substr(flags_table["fill_color"].rfin
d('.') + 1, std::string::npos - flags_table["fill_color"].rfind('.')));
    }
    else if (flags_table.find("fill") != flags_table.end())
    {
        std::cerr << "Where is fill_color?" << std::endl;
        exit(46);
    }
    else
    {
        circ.fill = 0;
    }

    return circ;
}

Ornament   orn_struct(std::unordered_map<std::string,   std::string>   &
flags_table)
{
    Ornament orn;

    orn.pattern = flags_table["pattern"];

    orn.count = std::stoi(flags_table["count"]);

    orn.thickness = std::stoi(flags_table["thickness"]);

    orn.color.b        =        std::stoi(flags_table["color"].substr(0,
flags_table["color"].find('.')));
    orn.color.g                                                          =
std::stoi(flags_table["color"].substr(flags_table["color"].find('.') + 1,
flags_table["color"].rfind('.') - flags_table["color"].find('.')));
    orn.color.r                                                          =
std::stoi(flags_table["color"].substr(flags_table["color"].rfind('.')   +
1, std::string::npos - flags_table["color"].rfind('.')));

    return orn;
}

Rotate    rot_struct(std::unordered_map<std::string,    std::string>    &
flags_table)
```

```
{
    Rotate rot;
    rot.left_up[0]      =      std::stoi(flags_table["left_up"].substr(0,
flags_table["left_up"].find('.')));
    rot.left_up[1]                                                    =
std::stoi(flags_table["left_up"].substr(flags_table["left_up"].find('.')
+ 1, std::string::npos - flags_table["left_up"].find('.')));

    rot.right_down[0]   =   std::stoi(flags_table["right_down"].substr(0,
flags_table["right_down"].find('.')));
    rot.right_down[1]                                                 =
std::stoi(flags_table["right_down"].substr(flags_table["right_down"].find
('.') + 1, std::string::npos - flags_table["right_down"].find('.')));

    rot.angle = std::stoi(flags_table["angle"]);

    return rot;
}
```

struct_work.h

```
#pragma once

#include "structs.h"
#include <iostream>
#include <unordered_map>
#include "../image_methods/one_class.h"


Rectangle   rect_struct(std::unordered_map<std::string,   std::string>   &
flags_table);
Circle    circ_struct(std::unordered_map<std::string,    std::string>    &
flags_table);
Ornament    orn_struct(std::unordered_map<std::string,    std::string>    &
flags_table);
Rotate    rot_struct(std::unordered_map<std::string,    std::string>    &
flags_table);
```

structs.h

```
#pragma once

#include <array>
#include <string>
#include <cstdint>




#pragma pack(push, 1)
```

```
struct FileHeader {
    uint16_t type;
    uint32_t size;
    uint16_t reversed1;
    uint16_t reversed2;
    uint32_t Offset;
};

struct InfoHeader {
    uint32_t headerSize;
    int32_t width;
    int32_t height;
    uint16_t colorPlanes;
    uint16_t bytesPerPixel;
    uint32_t compression;
    uint32_t imageSize;
    int32_t horizontalResolution;
    int32_t verticalResolution;
    uint32_t colorsUsed;
    uint32_t importantColors;
};

struct Color
{
    uint8_t r, g, b;

    Color()
        : b(0), g(0), r(0) {}
    Color(uint8_t, uint8_t g, uint8_t b)
        : b(b), g(g), r(r) {}
    int operator!= (Color& other)
    {
        if (r != other.r || b != other.b || g != other.g)
        {
            return 1;
        }
        return 0;
    }
    int operator== (Color& other)
    {
        if (r == other.r && b == other.b && g == other.g)
        {
            return 1;
        }
        return 0;
    }
    void operator= (Color& other)
    {
        r = other.r;
        g = other.g;
```

```
        b = other.b;
    }
};

#pragma pack(pop)



struct Rectangle
{
    std::array<int, 2> left_up;
    std::array<int, 2> right_down;
    int thickness;
    Color color;
    bool fill;
    Color fill_color;
};

struct Ornament
{
    std::string pattern; //изменить в дальнейшем
    Color color;
    int thickness;
    int count;
};

struct Rotate
{
    std::array<int, 2> left_up;
    std::array<int, 2> right_down;
    int angle;
};

struct Circle
{
    std::array<int, 2> center;
    int radius;
    int thickness;
    Color color;
    bool fill;
    Color fill_color;
    Circle(std::array<int, 2> center, int radius, int thickness, Color
color, bool fill, Color fill_color)
    : center(center), thickness(thickness), radius(radius), color(color),
fill(fill), fill_color(fill_color) {}
    Circle()
    : center({0, 0}), thickness(0), radius(0), color({0,0,0}), fill(0),
fill_color({0,0,0}) {}
};
```

one_class.h

```
#pragma once

#include <string>
#include "../other/structs.h"
#include <vector>
#include <iostream>

class Image
{
public:
    Image()
        : bm_data(0), bm_width(0), bm_height(0) {}

    void readBMP(std::string & path, std::string & function_to_run);
    void exportBMP(std::string & path);

    int check_cords(int x, int y)
    {
        if (x < bm_width && y < bm_height && y >= 0 && x >= 0)
        {
            return 1;
        }
        return 0;
    }
    void rectangle(Rectangle & rect);
    void circle(Circle & circ);
    void rotate(Rotate & rot);
    void ornament(Ornament & orn);
    void line(int x1, int y1, int x2, int y2, Color color, int
thickness);

    unsigned long width() {return bm_width;}
    unsigned long height() {return bm_width;}
private:
    std::vector<std::vector<Color>> bm_data;


    FileHeader fileHeader;
    InfoHeader infoHeader;

    uint32_t bm_width;
    uint32_t bm_height;


};
```

io.cpp

```
#include "one_class.h"
#include <fstream>
#include "../other/messages.h"
#include "../other/structs.h"

void Image::readBMP(std::string & path, std::string & function_to_run) {
    std::ifstream f(path, std::ios::binary);
    if (!f.is_open()) {
        std::cerr << "Failed to open the input file" << std::endl;
        exit(44);
    }

    f.read(reinterpret_cast<char*>(&fileHeader), sizeof(fileHeader));
    f.read(reinterpret_cast<char*>(&infoHeader), sizeof(InfoHeader));

    if (fileHeader.type != 0x4D42 || infoHeader.bytesPerPixel != 24 ||
infoHeader.compression != 0) {
        std::cerr << WRONG_FILE << std::endl;
        exit(41);
    }



    bm_height = abs(infoHeader.height);
    bm_width = abs(infoHeader.width);


    if (function_to_run == "info")
    {
        std::cout << "File info:\n\
        1) file header size 14\n\
        2) info header size 40\n\
        3) bits per pixel " << infoHeader.bytesPerPixel << "\n\
        4) compression " << infoHeader.compression << "\n\
        5) Width " << bm_width << "\n\
        6) Height " << bm_height << std::endl;
        exit(0);
    }

    bm_data.resize(bm_height);
    for (int y = 0; y < bm_height; y++)
    {
        bm_data[y].resize(bm_width);
    }

    const int padding = ((4 - (bm_width * 3) % 4) % 4);

    for (int y = 0; y < bm_height; y++) {
        for (int x = 0; x < bm_width; x++) {
            uint8_t pixel[3];
```

```
            f.read(reinterpret_cast<char*>(pixel), 3);
            bm_data[y][x].r = pixel[0];
            bm_data[y][x].g = pixel[1];
            bm_data[y][x].b = pixel[2];
        }
        f.ignore(padding);
    }
    f.close();

    for (int i = 0; i < bm_data.size() / 2; i++)
    {
        std::swap(bm_data[i], bm_data[bm_data.size()-i-1]);
    }

}

void Image::exportBMP(std::string & path) {
    for (int i = 0; i < bm_data.size() / 2; i++)
    {
        std::swap(bm_data[i], bm_data[bm_data.size()-i-1]);
    }

    std::ofstream f;
    f.open(path, std::ios::out | std::ios::binary);

    if (!f.is_open())
    {
        std::cerr << "Failed to open the output file" << std::endl;
        exit(44);
    }

    f.write(reinterpret_cast<char*>(&fileHeader), sizeof(fileHeader));
    f.write(reinterpret_cast<char*>(&infoHeader), sizeof(infoHeader));

    const int padding = ((4 - (bm_width * 3) % 4) % 4);

    unsigned char bmpPad[3]{0, 0, 0};
    for(int y = 0; y < bm_height; y++)
    {
        for (int x = 0; x < bm_width; x++)
        {
            unsigned char pixel[3];
            pixel[0] = bm_data[y][x].r;
            pixel[1] = bm_data[y][x].g;
            pixel[2] = bm_data[y][x].b;
            f.write(reinterpret_cast<char*>(pixel), 3);
        }
        f.write(reinterpret_cast<char*>(bmpPad), padding);
    }
    f.close();
```

```
}
```

image_work.cpp

```
#include "one_class.h"
#include <unordered_map>
#include "../other/messages.h"
#include <cmath>


void Image::line(int x1, int y1, int x2, int y2, Color color, int
thickness)
{
    const int deltaX = abs(x2 - x1);
    const int deltaY = abs(y2 - y1);
    const int signX = x1 < x2 ? 1 : -1;
    const int signY = y1 < y2 ? 1 : -1;
    int error = deltaX - deltaY;
    Circle circ = {{x2, y2}, thickness / 2, 1, color, false, color};
    circle(circ);

    while(x1 != x2 || y1 != y2)
   {
        circ.center = {x1, y1};
        circle(circ);
        int error2 = error * 2;
        if(error2 > -deltaY)
        {
            error -= deltaY;
            x1 += signX;
        }
        if(error2 < deltaX)
        {
            error += deltaX;
            y1 += signY;
        }
    }
}

void Image::rectangle(Rectangle & rect)
{
    int x1 = rect.left_up[0] < rect.right_down[0] ? rect.left_up[0] :
rect.right_down[0];
    int x2 = rect.left_up[0] < rect.right_down[0] ? rect.right_down[0] :
rect.left_up[0];
    int y1 = rect.left_up[1] < rect.right_down[1] ? rect.left_up[1] :
rect.right_down[1];
    int y2 = rect.left_up[1] < rect.right_down[1] ? rect.right_down[1] :
rect.left_up[1];
```

```
    for (int y = y1 + rect.thickness / 2; y <= y2 - rect.thickness / 2;
y++)
    {
        for (int x = x1 + rect.thickness / 2; x <= x2 - rect.thickness /
2; x++)
        {
            if (y >= 0 && y < bm_height && x >= 0 && x < bm_width)
            {
                bm_data[y][x] = rect.fill_color;
            }
        }
    }

    line(x1, y1, x1, y2, rect.color, rect.thickness); //право
    line(x2, y1, x2, y2, rect.color, rect.thickness); //лево
    line(x1, y1, x2, y1, rect.color, rect.thickness); //верх
    line(x1, y2, x2, y2, rect.color, rect.thickness); //низ


}

void Image::circle(Circle & circ)
{
    int xc = circ.center[0];
    int yc = circ.center[1];
    int x1 = xc - circ.radius, x2 = xc + circ.radius, y1 = yc -
circ.radius, y2 = yc + circ.radius;

    if (circ.fill == 1)
    {
        for (int y = y1; y <= y2; y++)
        {
            for (int x = x1; x <= x2; x++)
            {
                if ((y - yc)*(y - yc) + (x - xc)*(x - xc) <=
circ.radius*circ.radius && check_cords(x, y)) {
                    bm_data[y][x] = circ.fill_color;
                }
            }
        }
    }

    for (int k = -circ.thickness / 2 - circ.thickness % 2; k <=
circ.thickness / 2 + circ.thickness % 2; k++)
    {
        int x = 0;
        int r = circ.radius + k;
        int y = r;
        int d = 1 - 2 * r;
        int error = 0;
```

```
        while (y >= 0) {
            if (check_cords(xc + x, yc + y)) //вторая четверть
            {
                bm_data[yc + y][xc + x] = circ.color;
            }
            if (check_cords(xc + x, yc + y - 1) && (check_cords(xc + x,
yc + y - 2) && bm_data[yc + y - 2][xc + x] == circ.color))
            {
                bm_data[yc + y - 1][xc + x] = circ.color;
            }
            if (check_cords(xc + x, yc - y)) //первая четверть
            {
                bm_data[yc - y][xc + x] = circ.color;
            }
            if (check_cords(xc + x, yc - y + 1) && check_cords(xc + x, yc
- y + 2) && bm_data[yc - y + 2][xc + x] == circ.color)
            {
                bm_data[yc - y + 1][xc + x] = circ.color;
            }
            if (check_cords(xc - x, yc + y)) //третья четрверть
            {
                bm_data[yc + y][xc - x] = circ.color;
            }
            if (check_cords(xc - x, yc + y - 1) && check_cords(xc - x, yc
+ y - 2) && bm_data[yc + y - 2][xc - x] == circ.color)
            {
                bm_data[yc + y - 1][xc - x] = circ.color;
            }
            if (check_cords(xc - x, yc - y)) //четвертая четверть
            {
                bm_data[yc - y][xc - x] = circ.color;
            }
            if (check_cords(xc - x, yc - y + 1) && (check_cords(xc - x,
yc - y + 2) && bm_data[yc - y + 2][xc - x] == circ.color))
            {
                bm_data[yc - y + 1][xc - x] = circ.color;
            }
            error = 2 * (d + y) - 1;
            if (d < 0 && error <= 0)
            {
                ++x;
                d += 2 * x + 1;
                continue;
            }
            if(d > 0 && error > 0) {
                --y;
                d += 1 - 2 * y;
                continue;
              }
```

```
            ++x;
            d += 2 * (x - y);
            --y;
        }

    }

}


void Image::rotate (Rotate & rot)
{
    rot.left_up[0] = rot.left_up[0] >= bm_width ? bm_width - 1 :
rot.left_up[0];
    rot.left_up[1] = rot.left_up[1] >= bm_height ? bm_height - 1 :
rot.left_up[1];
    rot.right_down[0] = rot.right_down[0] >= bm_width ? bm_width - 1 :
rot.right_down[0];
    rot.right_down[1] = rot.right_down[1] >= bm_height ? bm_height - 1 :
rot.right_down[1];

    if (rot.left_up[0] > rot.right_down[0])
    {
        int tmp = rot.left_up[0];
        rot.left_up[0] = rot.right_down[0];
        rot.right_down[0] = tmp;
    }
    if (rot.left_up[1] > rot.right_down[1])
    {
        int tmp = rot.left_up[1];
        rot.left_up[1] = rot.right_down[1];
        rot.right_down[1] = tmp;
    }
    int x1 = rot.left_up[0];
    int y1 = rot.left_up[1];
    int x2 = rot.right_down[0];
    int y2 = rot.right_down[1];


    int h = y2 - y1;
    int w = x2 - x1;


    int p1 = abs(w - h) % 2 == 0 ? abs(w - h) / 2 : (abs(w - h) / 2) + 1;

    std::vector<std::vector<Color>> temp(w, std::vector<Color>(h));
    if (rot.angle == 90)
    {
        int xn, yn;
```

```
    xn = w >= h ? p1 + x1 : x1 - p1;
    yn = w >= h ? y1 - p1 : y1 + p1;

    if (w % 2 == 1 && h % 2 == 0 && w > h)
    {
        xn--;
        yn++;
    }
    else if (w % 2 == 0 && h % 2 == 1 && h > w)
    {
        xn++;
        yn--;
    }


    for (int i = 0; i < w; i++)
    {
        for (int j = 0; j < h; j++)
        {
            if (check_cords(x2 - i - 1, y1 + j))
            {
                temp[i][j] = bm_data[y1 + j][x2 - i - 1];
            }
        }
    }
    for (int i = 0; i < w; i++)
    {
        for (int j = 0; j < h; j++)
        {

            if (check_cords(xn + j, yn + i))
            {
                bm_data[yn+i][xn+j] = temp[i][j];
            }
        }
    }


}
else if (rot.angle == 270)
{
    int xn, yn;

    xn = w >= h ? p1 + x1 : x1 - p1;
    yn = w >= h ? y1 - p1 : y1 + p1;

    if (w % 2 == 1 && h % 2 == 0 && w > h)
    {
        xn--;
        yn++;
```

```
    }
    else if (w % 2 == 0 && h % 2 == 1 && h > w)
    {
        xn++;
        yn--;
    }

    for (int i = 0; i < w; i++)
    {
        for (int j = 0; j < h; j++)
        {
            if (check_cords(x1 + i, y2 - j - 1))
            {
                temp[i][j] = bm_data[y2 - j - 1][x1 + i];
            }
        }
    }

    for (int i = 0; i < w; i++)
    {
        for (int j = 0; j < h; j++)
        {

            if (check_cords(xn + j, yn + i))
            {
                bm_data[yn + i][xn + j] = temp[i][j];
            }
        }
    }

}
else if (rot.angle == 180)
{
    std::vector<std::vector<Color>> tmp(h, std::vector<Color>(w));
    for (int y = 0; y < h; y++)
    {
        for (int x = 0; x < w; x++)
        {
            if (check_cords(x2 - x - 1, y1 + y))
            {
                tmp[y][x] = bm_data[y1 + y][x2 - x - 1];
            }
        }
    }
    for (int y = 0; y < tmp.size() / 2; y++)
    {
        std::swap(tmp[y], tmp[tmp.size() - y - 1]);
    }
    for (int y = 0; y < h; y++)
    {
```

```
            for (int x = 0; x < w; x++)
            {
                if (check_cords(x1 + x, y1 + y))
                {
                    bm_data[y1 + y][x1 + x] = tmp[y][x];
                }
            }
        }
    }
}

void Image::ornament(Ornament & orn)
{
    if (orn.pattern == "circle")
    {
        double xc = bm_width / 2;
        double yc = bm_height / 2;
        double rad = yc > xc ? xc : yc;

        std::array<double, 2> center{xc, yc};
        for (int y = 0; y < bm_height; y++)
        {
            for (int x = 0; x < bm_width; x++)
            {
                double hyp = pow(pow(center[0] - x, 2.0) + pow(center[1]
- y, 2.0), 0.5);
                if (hyp > rad)
                {
                    bm_data[y][x] = orn.color;
                }
            }
        }
    }
    else if (orn.pattern == "rectangle")
    {
        Rectangle    rect{{0,    static_cast<int>(bm_height    -    1)},
{static_cast<int>(bm_width - 1), 0}, orn.thickness, orn.color, false};
        int y1 = -1;
        int x1 = -1;
        int y2 = bm_height;
        int x2 = bm_width;
        for (int k = 0; k < orn.count; k++)
        {
            for (int i = 0; i < orn.thickness; i++)
            {
                y1++, x1++, y2--, x2--;
                for (int y = y1; y <= y2; y++) {
                    if (check_cords(x1, y)){
                        bm_data[y][x1] = rect.color; //лево
                    }
```

```
                    if (check_cords(x2, y)){
                        bm_data[y][x2] = rect.color; //право
                    }
                }


                for (int x = x1; x <= x2; x++) {
                    if (check_cords(x, y1))
                    {
                        bm_data[y1][x] = rect.color; //верх
                    }
                    if (check_cords(x, y2)) {
                        bm_data[y2][x] = rect.color; //низ
                    }
                }
            }

            x1 += orn.thickness;
            y1 += orn.thickness;
            x2 -= orn.thickness;
            y2 -= orn.thickness;
            if (x2 < 0 || y2 < 0)
            {
                std::cerr << "Oops, too much rectangles, but it's ok" <<
std::endl;
                break;
            }
        }
    }

    else if (orn.pattern == "semicircles")
    {
        int wd = bm_width / orn.count;
        int hd = bm_height / orn.count;
        Circle circ({wd / 2, static_cast<int>(bm_height - 1)}, wd / 2,
orn.thickness, orn.color, false, {0,0,0});
        for (int k = 0; k < orn.count; k++)
        {
            circle(circ);
            circ.center[0] += wd + 1;
        }
        circ.center = {wd / 2, 0};
        for (int k = 0; k < orn.count; k++)
        {
            circle(circ);
            circ.center[0] += wd + 1;
        }
        circ.center = {0, hd / 2};
        circ.radius = hd / 2;
```

```
            for (int k = 0; k < orn.count; k++)
            {
                circle(circ);
                circ.center[1] += hd + 1;
            }
            circ.center = {static_cast<int>(bm_width - 1), hd / 2};
            for (int k = 0; k < orn.count; k++)
            {
                circle(circ);
                circ.center[1] += hd + 1;

            }
        }
    }
```