МИНОБРНАУКИ РОССИИ
Санкт-Петербургский государственный
электротехнический университет
«ЛЭТИ» им. В. И. Ульянова (Ленина)
КАФЕДРА МОЕВМ

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка изображений

Студент гр. 3344 _____ **Вердин К.К.**

Преподаватель _____ **Глазунов С.А.**

Санкт-Петербург
2024

ЗАДАНИЕ

на курсовую работу

Студент Вердин К.К.

Группа 3344

Тема работы: Обработка изображений.

Исходные данные:

- Программа **обязательно должна иметь CLI** (опционально дополнительное использование GUI).
- Программа должна реализовывать весь следующий функционал по обработке bmp-файла
- 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).
- Каждую подзадачу следует вынести в отдельную функцию, функции сгруппировать в несколько файлов
- Сборка должна осуществляться при помощи make и Makefile или другой системы сборки

Содержание пояснительной записки:

- Содержание
- Введение
- Описание задания
- Описание реализованных функций, структур
- Описание файловой структуры программы
- Описание модульной структуры, сборки программы
- Примеры работы программы
- Примеры ошибок
- Заключение
- Список использованных источников
- Приложение А. Исходный код программы

Предполагаемый объем пояснительной записки:

Не менее 30 страниц.

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 22.05.2024

Дата защиты реферата: 22.05.2024

Студент _____ Вердин К.К.

Преподаватель _____ Глазунов С.А.

Аннотация

Курсовая работа подразумевает написание программы, которая обрабатывает bmp-файл с использованием CLI интерфейса. Программа производит считывание и обработку текстовых файлов по заданным в командной строке флагам и параметрам. При написании программы использовались методы работы с изображением, структурами, динамической памятью и функциями стандартной библиотеки. Обработка bmp-файлов включает в себя 4 функции обработки изображения (нахождение всех прямоугольников заданного цвета, рисование окружности, фильтр rgb-компонента, Разделение изображения на N*M частей). Результатом работы программы является обработанное изображение, которое будет сохранено в файл с заданным именем. Также результатом работы программы может быть справка о реализованных внутри программы функциях, полная информация о считанном bmp-файле или же ошибка с указанием на её причину.

содержание

Введение

Целью данной работы является создание программы на языке программирования C++, которая будет обрабатывать BMP-изображение с помощью CLI интерфейса.

Для достижения поставленной цели требуется решить следующие задачи:

1. Изучить формат BMP
2. Изучить методы реализации CLI интерфейса
3. Реализовать функции обработки изображения
4. Реализовать эффективную сборку программы
5. Предусмотреть возможные ошибки и их причины

Возможные методы решения поставленных задач:

1. Реализация класса и методов для считывания, записи и обработки bmp-файлов
2. Реализация структур-хедеров для считанного изображения
3. Использование библиотеки getopt для работы с командной строкой
4. Сборка проекта с помощью Makefile
5. Вынесение каждой подзадачи в отдельную функцию

1. ОПИСАНИЕ ЗАДАНИЯ

Программа должна иметь следующую функции по обработке изображений:

1. Поиск всех залитых прямоугольников заданного цвета. Флаг для выполнения данной операции: `--filled_rects`. Требуется найти все прямоугольники заданного цвета и обвести их линией. Функционал определяется:

Цветом искомых прямоугольников. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)

Цветом линии для обводки. Флаг `--border_color` (работает аналогично флагу `--color`)

Толщиной линии для обводки. Флаг `--thickness`. На вход принимает число больше 0

2.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`)

3.Фильтр rgb-компонент. Флаг для выполнения данной операции: `--rgbfilter`. Этот инструмент должен позволять для всего изображения либо установить в диапазоне от 0 до 255 значение заданной компоненты. Функционал определяется

Какую компоненту требуется изменить. Флаг `--component_name`. Возможные значения `red`, `green` и `blue`.

В какой значение ее требуется изменить. Флаг `--component_value`. Принимает значение в виде числа от 0 до 255

4.Разделяет изображение на N*M частей. Флаг для выполнения данной операции: `--split`. Реализация: провести линии заданной толщины. Функционал определяется:

Количество частей по “оси” Y. Флаг `--number_x`. На вход принимает число больше 1

Количество частей по “оси” X. Флаг `--number_y`. На вход принимает число больше 1

Толщина линии. Флаг `--thickness`. На вход принимает число больше 0

Цвет линии. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)

2. ОПИСАНИЕ ПРОГРАММЫ

2.1. Реализованные функции, структуры

Во время разработки программы был реализован класс BMP со следующими полями и методами:

Во время разработки программы были реализованы структуры:

1. Rgb - используется для хранения цвета пикселя. Для данной структуры были реализованы перегрузка операторов == и *!=*

2. BitmapFileHeader - используется для хранения общей информации об изображении.

3. BitmapInfoHeader - используется для хранения подробной информации об изображении и определения формата пикселей.

4. Coordinates - используется для хранения координат конкретной точки.

5. Circle - используется для хранения данных о круге.

6.input_data - используется для хранения данных, которые были введены пользователем.

7. struct option- структура библиотеки getopt.

Во время разработки программы был реализован класс BMP со следующими полями и методами:

Поля:

BitmapInfoHeader info_header - хранит основную информацию об изображении

BitMapFileHeader file_header - хранит информацию о цветах изображения;

std::vector<std::vector<RGB>> pixels - хранит информацию о цвете каждого пикселя изображения

int padding_size - хранит информацию о размере выравнивания.

Методы:

*void BMP::read(const char *input_path)* - считывает изображение по заданному пути к файлу

void BMP::read_headers(std::ifstream &bmp_file) - считывает заголовок файла и заголовок DIB

void BMP::read_pixels(std::ifstream &bmp_file) - считывает пиксели изображения и записывает их в двухмерный массив

*void BMP::write(const char *output_path)* - записывает данные объекта класса BMP в файл по заданому пути

void BMP::write_headers(std::ofstream &out_file) - записывает данные заголовков в файл

void BMP::write_pixels(std::ofstream &out_file) - записывает данные пикселей в файл

void BMP::rgbfilter(std::string component_name, uint8_t component_value) - устанавливает для всего изображения значение заданной компоненты от 0 до 255

*void BMP::set_component_value_of_pixel(std::string component_name, uint8_t component_value, RGB *pixel)* - устанавливает для пикселя значение заданной компоненты от 0 до 255

void BMP::set_color_of_pixel(RGB color, int x, int y) - устанавливает для пикселя заданный цвет

void BMP::draw_circle_line(Circle c, RGB color) - рисует не залитую цветом окружность толщина границы которой равна 1

void BMP::fill_circle(Circle c, RGB color) - рисует залитую цветом окружность с толщиной линии 1

void BMP::draw_circle(Circle c, RGB color, int thickness) - рисует окружность заданной толщины без заливки

void BMP::draw_circle(Circle c, RGB color, int thickness, RGB fill_color) - рисует окружность заданной толщины с заливкой

void BMP::draw_line(Coordinates coordinates, int thickness, RGB color) - рисует линию, через две точки заданной толщины и цвета

std::vector<int> BMP::coordinates_split(int num, int len_axis) - возвращает массив координат для метода split

void BMP::split(int number_x, int number_y, int thickness, RGB color) - разделяет изображение на N*M частей, проводит линии заданной толщины и цвета

void BMP::change_pixels_of_field(RGB color, Coordinates coordinates) - изменяет цвет всех пикселей заданной области

std::vector<std::vector<int>> BMP::free_border_data(RGB color) - возвращает массив с информацией о количестве соседних пикселей, цвет которых не является заданным, для каждого пикселя.

void BMP::filled_rects(RGB color, int thickness, RGB border_color) - находит все прямоугольники заданного цвета и обводит их линиями заданной толщины и цвета.

void BMP::print_file_header() - выводит информацию о заголовочном файле

void BMP::print_info_header() - выводит информацию о DIB заголовке

Также во время разработки были реализованы функции для обработки пользовательского ввода:

*unordered_map<string, string> input(int argc, char **argv)* - возвращает хэш-таблицу с парами {флаг:аргумент переданный пользователем)

*input_data *check_input(unordered_map<string, string> input)* - проверяет аргументы на соответствие шаблонам и возвращает структуру с переданными пользователем параметрами

*vector<int> parse_input(string str, void(*func)(int))* - возвращает массив целых чисел полученных из строки

vector<int> parse_input(string str) - возвращает массив целых чисел полученных из строки

void check_rgb_value(int value) - проверяет на валидность значение rgb-компонента

vector<string> split(string str, char ch) - разделяет строку на строки по определённому символу

int custom_stoi(string str) - превращает строку в число

void check_positive(int val) - проверяет, является ли число положительным

void check_component_name(string str) - проверяет, соответствует ли переданная строка названию gb-компоненты

void print_help() - выводит справку о функционале курсовой работы

2.2. Файловая структура программы

Во время разработки программа была разбита на следующие файлы:

· Makefile - файл, необходимый для компиляции и сборки проекта.

· bmp.cpp - файл, содержащий функции считывания, записи, вывода информации о bmp-файле.

· bmp.h - заголовочный файл, содержащий прототипы функций считывания, записи, вывода информации о bmp-файле.

· input.cpp - файл Содержащий Функции обработки информации введённой пользователем

· input.h - заголовочный файл содержащий прототипы функций обработки введённой пользователем информации

· main.cpp

2.3. Модульная структура, сборка

Для сборки проекта используется Makefile:

· cw – исполняемый файл, требует все нижеперечисленные объектные файлы для сборки и линковки.

· bmp.o- объектный файл, требующий bmp/bmp.cpp bmp/bmp.h для компиляции.

· input.o- объектный файл, требующий bmp/bmp.cpp bmp/bmp.h для компиляции.

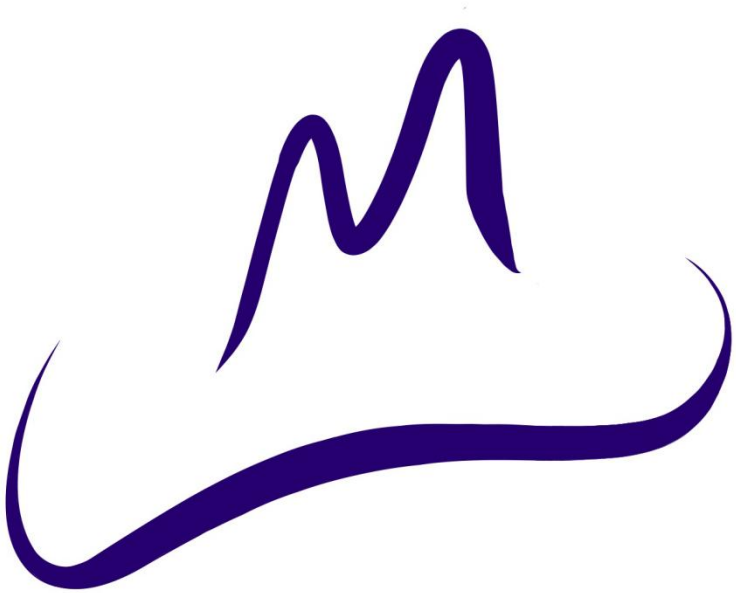
· main.o- объектный файл, требующий bmp/bmp.cpp bmp/bmp.h для компиляции.

· Clean - очистка всех объектных файлов и исполняемого файла cw.

Компиляция происходит с помощью: g++ -std=c++11.

3. ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

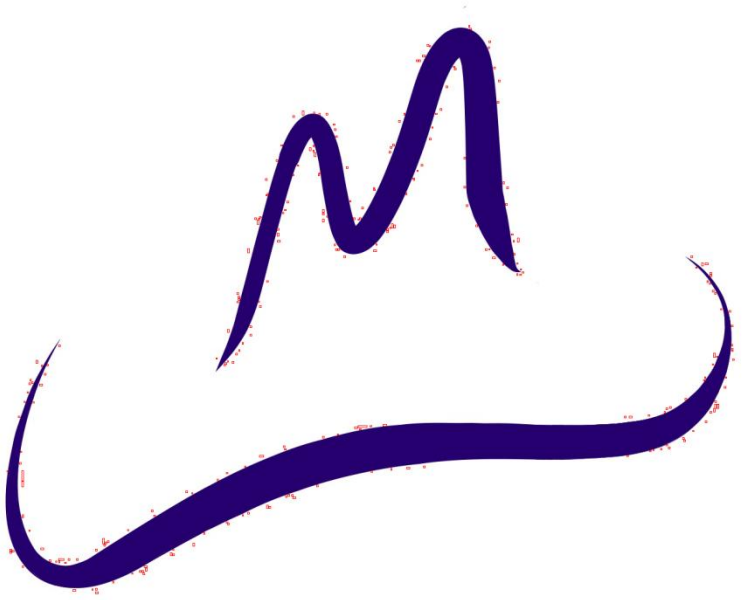
1.Нахождение всех прямоугольников заданного цвета



Исходная картинка(moevm_the_best_image.bmp):

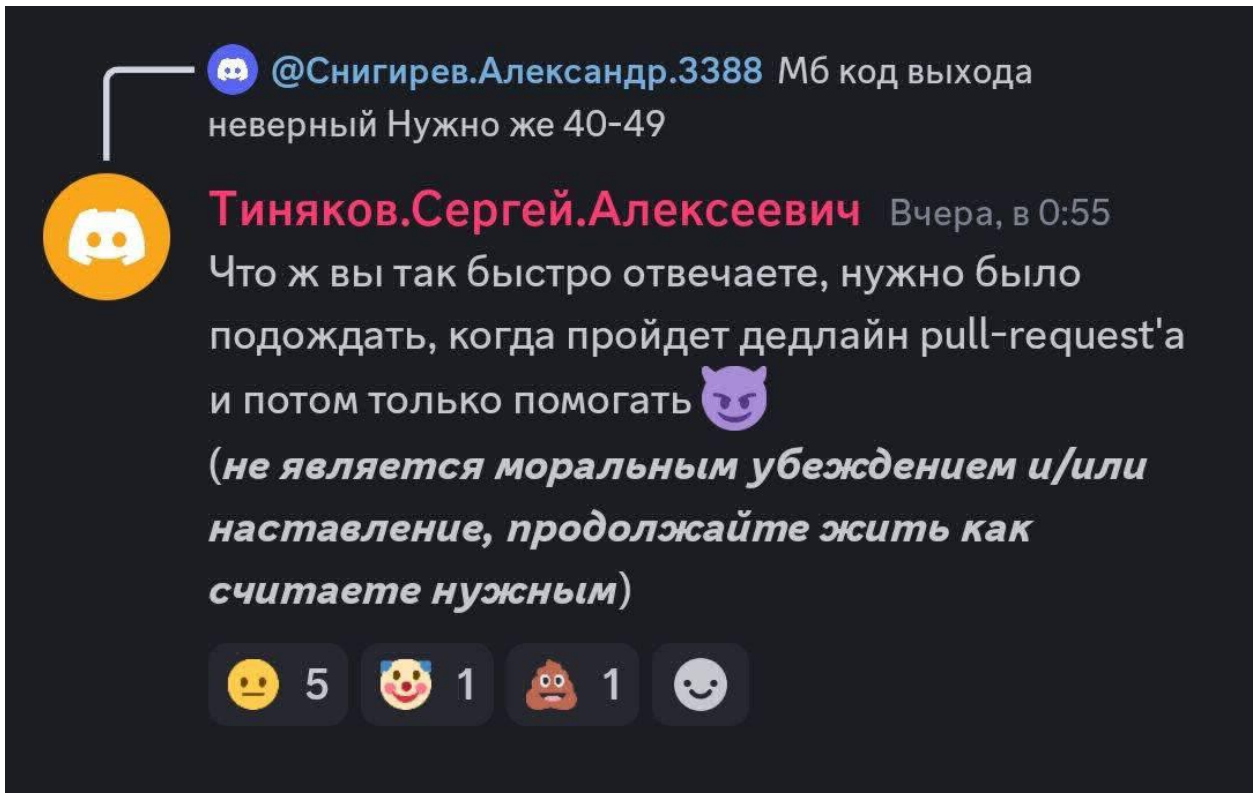
Входные данные:

./cw --input ../input_image_examples/moevm_image.bmp --filled_rects --color 255.255.255 --border_color 255.0.0 --thickness 1 --output ../output_image_examples/moevm_image_filledrects.bmp



Обработанное изображение:

2. Изменение значения rgb-компоненты:



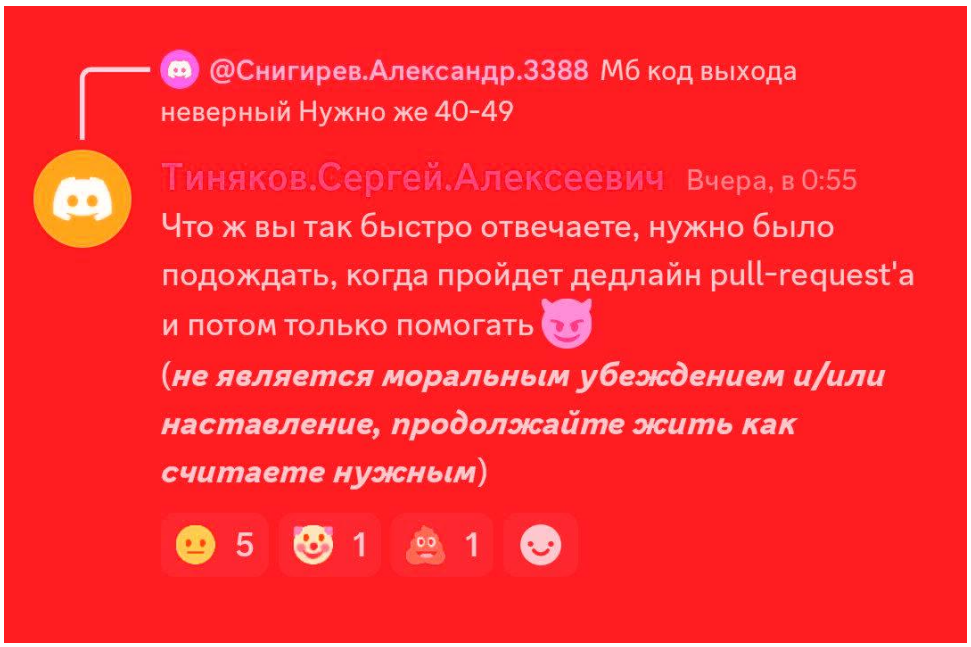
Исходное изображение Tinyakov_the_best_professor.bmp

Deleted[Кирилл Вердин]:

Входные данные:

./cw --input ../input_image_examples/Tinyakov_the_best_professor.bmp --rgbfilter --component_name red --component_value 255 --output ../output_image_examples/tinyakov_imge_rgbfilter.bmp

Обработанное изображение:



3. Рисование окружности

Исходное изображение Asya_the_prettiest_cat.bmp

Входные данные:

./cw --input ../input_image_examples/Asya_the_prettiest_cat.bmp --circle --center 900.150 --radius 100 --thickness 20 --color 107.85.115 --fill --fill_color 87.65.95 --output ../output_image_examples/asya_image_circle.bmp

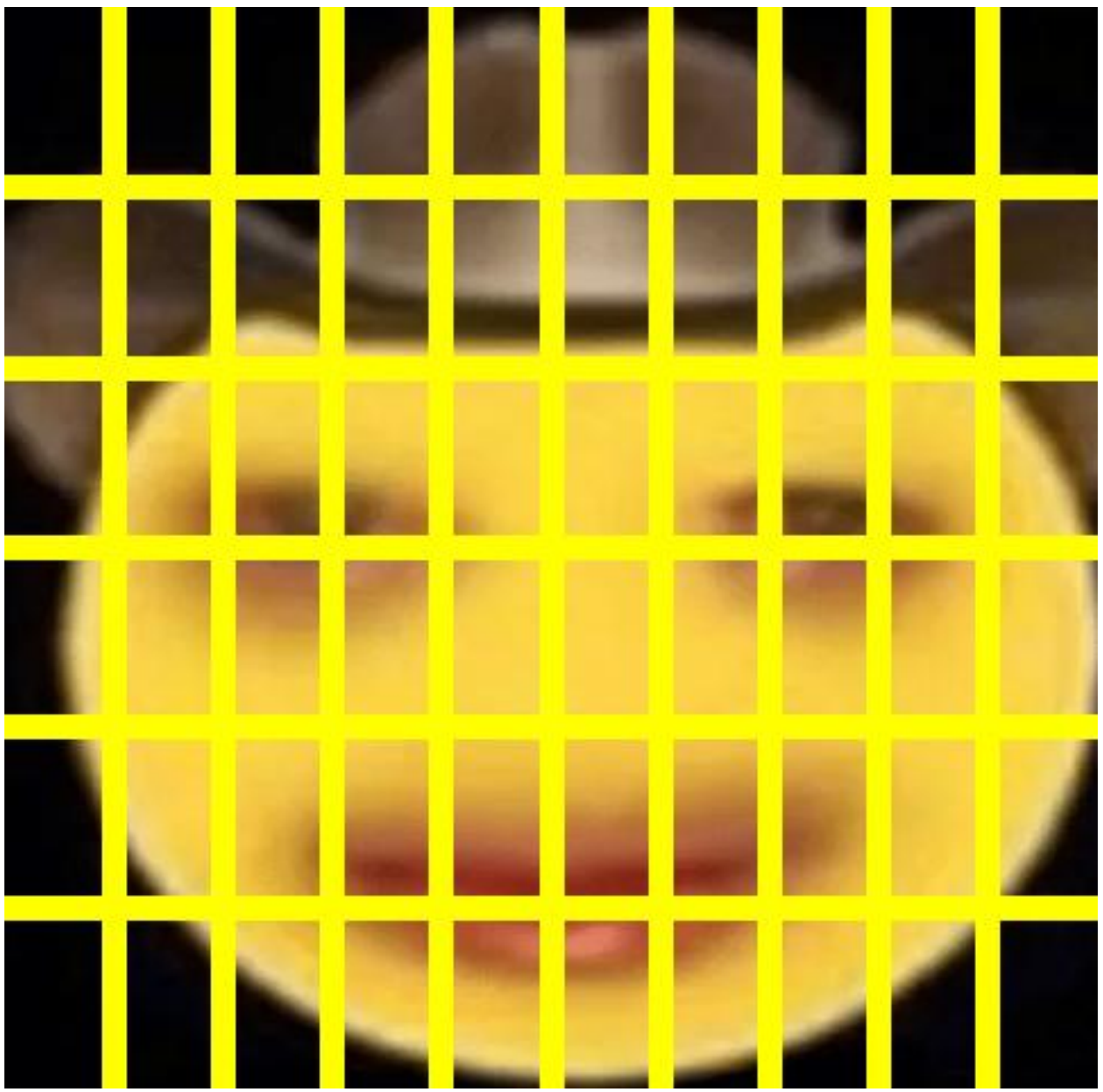
Обработанное изображение:

4. Разделение изображения на части

Исходное изображение very_sad_cowboy.bmp

Входные данные:

./cw --input ../input_image_examples/very_sad_cowboy.bmp --split --number_x 10 --number_y 6 --color 255.255.0 --thickness 10 --output ../output_image_examples/very_sad_cowboy_split.bmp



Вывод программы:

Вывод справки

Входные данные:

-help

Вывод программы

```
Course work for option 5.8, created by Kirill Verdin.
Options:
--input/-i - enter path of input file
--output/-o - enter path of output file

--info - print info about image

--help/-h - print list of functions

--filled_rects - find all rectangles of the certain color
--color <r.g.b> - color of rectangles
--border_color <r.g.b> - border_color of rectangles
--thickness - thickness of border line of rectange

--circle - draw circle
--center <x.y> - coordinates of the center of circle
--radius - radius of circle
--color <r.g.b> - color of border of circle
--thickness - thickness of border line
--fill - fill/not fill circle
--fill_color <r.g,b> color of circle

--rgbfilter - change the value of 1 component of color for whole image
--component_name <red/green/blue> - name of RGB component that would change
--component_value -- value of RGB component

--split - split image on x*y parts
--number_x - number of parts on axis y
--number_y - number of parts on axis x
-thickness - thickness of line
--color <r.g,b> - color of line
```

ЗАКЛЮЧЕНИЕ

Была успешно создана программа, которая обрабатывает изображение в зависимости от подаваемых пользователем флагов и аргументов в командную строку. Программа выполняет поставленные задачи по считыванию, обработке и записи BMP-изображений. При выполнении задания были улучшены навыки работы с изображением, также был получен опыт использование CLI интерфейса, реализованного при помощи библиотеки getopt. Полученные результаты показывают, что поставленные цели были успешно достигнуты.

список использованных источников

1. The GNU C Library Reference Manual. GETOPT. URL: https://www.gnu.org/software/libc/manual/html_node/Getopt.html (дата обращения 08.05.2024)

2. Бьёрн Страуструп: A Tour of C++ (2nd Edition) (2018) США: Addison-Wesley, 2018 г.

3. Базовые сведения к выполнению курсовой работы по дисциплине «программирование». второй семестр: учеб.-метод. Пособие сост. А. А. Лисс, С. А. Глазунов, М. М. Заславский, К. В. Чайка и др. СПб.: Изд-во СПбГЭТУ "ЛЭТИ", 2024. 36 с.

4. Язык программирования C++ упражнения и лекции 5-ое издание. Стивен Прага. Из. «Вильямс» 2007г

приложение А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Makefile

```
CC =g++
CFLAGS =-std=c++11

all: cw

cw: main.o bmp.o input.o
    ${CC} $^ -o $@ ${CFLAGS}

bmp.o: bmp/bmp.cpp bmp/bmp.h
    ${CC} -c $< ${CFLAGS}

input.o: input/input.cpp input/input.h bmp/bmp.h
    ${CC} -c $< ${CFLAGS}

main.o: main.cpp input/input.h bmp/bmp.h
    ${CC} -c $< ${CFLAGS}

clean:
    rm *.o cw
```

Bmp.cpp

```
#include "bmp.h"
#include <fstream>
#include <iostream>
#include <cmath>

std::ostream &operator<<(std::ostream &o, const RGB& color)
{
    o << unsigned(color.r) << '\t' << unsigned(color.g) << '\t' <<
unsigned(color.b) << '\n';
    return o;
}

BMP::BMP()
{

}

BMP::BMP(const char *input_path)
{
    read(input_path);
}

void BMP::read(const char *input_path)
{
    std::ifstream bmp_file{input_path, std::ios_base::binary};

    if (!bmp_file)
    {
```

```
        std ::cout << "the file could not be opened or the file path
was specified incorrectly\n";
        exit(44);
    }
    read_headers(bmp_file);
    if (file_header.signature != 0x4D42 || info_header.bitsPerPixel !=
24 || info_header.compression != 0)
    {
        std::cerr << "Wrong file\n";
        exit(45);
    }
    read_pixels(bmp_file);
    bmp_file.close();
}

void BMP::read_headers(std::ifstream &bmp_file)
{
    bmp_file.read((char *)&file_header, sizeof(BitMapFileHeader));
    bmp_file.read((char *)&info_header, sizeof(BitmapInfoHeader));
}

void BMP::read_pixels(std::ifstream &bmp_file)
{
    padding_size = (info_header.width * sizeof(RGB)) % 4;
    if (padding_size)
        padding_size = 4 - padding_size;
    bmp_file.seekg(file_header.pixelArrOffset, bmp_file.beg);
    unsigned int H = info_header.height;
    unsigned int W = info_header.width;
    std::vector<std::vector<RGB>>       pixels_data(info_header.height,
std::vector<RGB>(info_header.width));
    for (size_t i = 0; i < H; i++)
    {
        bmp_file.read((char*)(pixels_data[H  -  i  -  1].data()),  W  *
sizeof(RGB) + padding_size);
    }
    pixels = pixels_data;
}

void BMP::write(const char *output_path)
{
    std::ofstream output_file{output_path, std::ios_base::binary};

    write_headers(output_file);
    write_pixels(output_file);
    output_file.close();
}

void BMP::write_headers(std::ofstream &out_file)
{
    out_file.write((char *)&file_header, sizeof(BitMapFileHeader));
    out_file.write((char *)&info_header, sizeof(BitmapInfoHeader));
}

void BMP::write_pixels(std::ofstream &out_file)
{
    unsigned int H = info_header.height;
    unsigned int W = info_header.width;
```

```
    for (size_t i = 0; i < H; i++)
    {
        out_file.write((char  *)pixels[H  -  i  -  1].data(),  (W  *
sizeof(RGB)) + padding_size);
    }
}

void      BMP::rgbfilter(std::string      component_name,      uint8_t
component_value)
{
    for (size_t y = 0; y < info_header.height; y++)
        for (size_t x = 0; x < info_header.width; x++)
            set_component_value_of_pixel(component_name,
component_value, &pixels[y][x]);
}

void   BMP::set_component_value_of_pixel(std::string   component_name,
uint8_t component_value, RGB *pixel)
{
    if (component_name == "red")
        pixel->r = component_value;
    else if (component_name == "green")
        pixel->g = component_value;
    else if (component_name == "blue")
        pixel->b = component_value;
}

void BMP::draw_circle(Circle c, RGB color, int thickness)
{
    if (thickness == 1)
    {
        draw_circle_line(c, color);
        return;
    }
    int x = 0;
    int y = c.r;
    int d = 1 - 2 * c.r;
    int error = 0;
    int  radius_of_fill_circle  =  thickness  %  2  ?  thickness  /  2  :
(thickness + 1) / 2;
    while (y >= x)
    {
        fill_circle({c.x0  +  x,  c.y0  +  y,  radius_of_fill_circle},
color);
        fill_circle({c.x0  -  x,  c.y0  +  y,  radius_of_fill_circle},
color);
        fill_circle({c.x0  +  x,  c.y0  -  y,  radius_of_fill_circle},
color);
        fill_circle({c.x0  -  x,  c.y0  -  y,  radius_of_fill_circle},
color);
        fill_circle({c.x0  +  y,  c.y0  +  x,  radius_of_fill_circle},
color);
        fill_circle({c.x0  -  y,  c.y0  +  x,  radius_of_fill_circle},
color);
        fill_circle({c.x0  +  y,  c.y0  -  x,  radius_of_fill_circle},
color);
        fill_circle({c.x0  -  y,  c.y0  -  x,  radius_of_fill_circle},
color);
```

```
        error = 2 * (d + y) - 1;

        if (d < 0 && error <= 0)
            d += 2 * ++x + 1;
        else if (d > 0 && error >= 0)
            d -= 2 * --y + 1;
        else
            d += 2 * (++x - --y);
    }
}

void BMP::draw_circle(Circle c, RGB color, int thickness, RGB
fill_color)
{
    fill_circle(c, fill_color);
    draw_circle(c, color, thickness);
}

void BMP::draw_line(Coordinates coordinates, int thickness, RGB color)
{
    int dx = abs(coordinates.x1 - coordinates.x0);
    int dy = abs(coordinates.y1 - coordinates.y0);
    int sx = coordinates.x0 < coordinates.x1 ? 1 : -1;
    int sy = coordinates.y0 < coordinates.y1 ? 1 : -1;
    int err = dx - dy;
    int e2;
    for (int i = thickness / 2; i > 0; i--)
    {
        fill_circle({coordinates.x1, coordinates.y1, i}, color);
    }
    set_color_of_pixel(color, coordinates.x1, coordinates.y1);
    while (coordinates.x0 != coordinates.x1 || coordinates.y0 !=
coordinates.y1)
    {
        for (int i = thickness / 2; i > 0; i--)
        {
            fill_circle({coordinates.x0, coordinates.y0, i}, color);
        }
        set_color_of_pixel(color, coordinates.x0, coordinates.y0);
        e2 = 2 * err;
        if (e2 > -dy)
        {
            err -= dy;
            coordinates.x0 += sx;
        }
        if (e2 < dx)
        {
            err += dx;
            coordinates.y0 += sy;
        }
    }
}

void BMP::split(int number_x, int number_y, int thickness, RGB color)
{
    if (number_x*thickness >= info_header.width || number_y*thickness
>=info_header.height)
    {
```

```
        change_pixels_of_field(color,  {0,  0,  int(info_header.width),
int(info_header.height)});
        return;
    }

    std::vector<int>   coordinates_x   =   coordinates_split(number_x,
info_header.width);
    std::vector<int>   coordinates_y   =   coordinates_split(number_y,
info_header.height);
    for (auto v : coordinates_x)
        draw_line({v,  0,  v,  int(info_header.height)  -  1},  thickness,
color);
    for (auto v : coordinates_y)
        draw_line({0,  v,  int(info_header.width)  -  1,  v},  thickness,
color);
}

std::vector<int> BMP::coordinates_split(int num, int len_axis)
{
    std::vector<int> ans;
    for (size_t i = 1; i < num; i++)
    {
        ans.push_back(round(float(len_axis - 1) * i / num));
    }
    return ans;
}

void BMP::change_pixels_of_field(RGB color, Coordinates coordinates)
{
    for (size_t y = coordinates.y0; y < coordinates.y1; y++)
        for (size_t x = coordinates.x0; x < coordinates.x1; x++)
        {
            set_color_of_pixel(color, x, y);
        }
}

void BMP::filled_rects(RGB color, int thickness, RGB border_color)
{
    std::vector<std::vector<int>>       free_border_data_array       =
free_border_data(color);
    std::vector<Coordinates *> coords;
    for (int y = 0; y < info_header.height; y++)
    {
        for (int x = 0; x < info_header.width; x++)
        {
            if (pixels[y][x] != color)
                continue;
            Coordinates *c = new Coordinates;
            if (free_border_data_array[y][x] == 8)
            {
                c->x0 = x;
                c->x1 = x;
                c->y0 = y;
                c->y1 = y;
                coords.push_back(c);
                continue;
            }
            if (free_border_data_array[y][x] == 7)
```

```
{
    bool flag = false;
    for (int x1 = x + 1; x1 < info_header.width; x1++)
    {
        if (free_border_data_array[y][x1] != 6)
        {
            if (free_border_data_array[y][x1] != 7)
            {
                break;
            }
            c->x1 = x1;
            c->y0 = y;
            c->y1 = y;
            c->x0 = x;
            flag = true;
        }
    }
    if (!flag)
        for (int y1 = y + 1; y1 < info_header.height; y1++)
        {
            if (free_border_data_array[y1][x] != 6)
            {
                if (free_border_data_array[y1][x] != 7)
                {
                    break;
                }
                c->x1 = x;
                c->y0 = y;
                c->y1 = y1;
                c->x0 = x;
                flag = true;
            }
        }
    if (flag)
        coords.push_back(c);
    else
    {
        delete c;
        continue;
    }
}
else if (free_border_data_array[y][x] == 5)
{
    bool flag = true;
    for (int x1 = x + 1; x1 < info_header.width; x1++)
    {
        if (free_border_data_array[y][x1] != 3)
        {
            if (free_border_data_array[y][x1] == 5)
            {
                c->x1 = x1;
                c->x0 = x;
            }
            else
            {
                flag = false;
            }
            break;
```

```
    }
}
for (int y1 = y + 1; y1 < info_header.height; y1++)
{
    if (free_border_data_array[y1][x] != 3)
    {
        if (free_border_data_array[y1][x] == 5)
        {
            c->y1 = y1;
            c->y0 = y;
        }
        else
        {
            flag = false;
        }
        break;
    }
}
if (c->y0 + 1 > c->y1 || c->x0 + 1 > c->x1)
    flag = false;
if (!flag)
    continue;
for (int x1 = c->x0 + 1; x1 <= c->x1; x1++)
{
    if (free_border_data_array[c->y1][x1] != 3)
    {
        if (free_border_data_array[c->y1][x1] != 5)
            flag = false;
    }
        }
        for (int y1 = c->y0 + 1; y1 <= c->y1; y1++)
        {
            if (free_border_data_array[y1][c->x1] != 3)
            {
                if (free_border_data_array[y1][c->x1] != 5)
                    flag = false;
            }
        }
        if (!flag)
            continue;
        for (int y1 = c->y0 + 1; y1 < c->y1; y1++)
        {
            for (int x1 = c->x0 + 1; x1 < c->x1; x1++)
                if (free_border_data_array[y1][x1])
                {
                    flag = false;
                    break;
                }
            if (!flag)
                break;
        }
        if (flag)
            coords.push_back(c);
        else
            delete c;
        continue;
    }
}
```

```
    }
    for (auto v : coords)
    {
        // std::cout << v->x0 << '\t' << v->y0 << '\t' << v->x1 <<
'\t' << v->y1 << '\n';
        draw_line({v->x0 - 1, v->y0 - 1, v->x1 + 1, v->y0 - 1},
thickness, border_color);
        draw_line({v->x0 - 1, v->y1 + 1, v->x1 + 1, v->y1 + 1},
thickness, border_color);
        draw_line({v->x0 - 1, v->y0 - 1, v->x0 - 1, v->y1 + 1},
thickness, border_color);
        draw_line({v->x1 + 1, v->y0 - 1, v->x1 + 1, v->y1 + 1},
thickness, border_color);
    }
}

void BMP::print_info_of_pixels()
{
    for (size_t y = 5; y < 10; y++)
    {
        for (size_t x = 0; x < info_header.width; x++)
        {
            std::cout   <<   unsigned(pixels[y][x].r)   <<   "\t"   <<
unsigned(pixels[y][x].g) << "\t" << unsigned(pixels[y][x].b) << '\n';
        }
    }
}

void BMP::set_color_of_pixel(RGB color, int x, int y)
{
    if (x < 0 || x >= info_header.width || y < 0 || y >=
info_header.height)
        return;
    if (color.r <= 255)
    {
        pixels[y][x].r = color.r;
    }
    if (color.b <= 255)
    {
        pixels[y][x].b = color.b;
    }
    if (color.g <= 255)
    {
        pixels[y][x].g = color.g;
    }
}

void BMP::draw_circle_line(Circle c, RGB color)
{
    int x = 0;
    int y = c.r;
    int d = 3 - 2 * y;
    while (y >= x)
    {
        set_color_of_pixel(color, c.x0 + x, c.y0 + y);
        set_color_of_pixel(color, c.x0 - x, c.y0 + y);
        set_color_of_pixel(color, c.x0 + x, c.y0 - y);
        set_color_of_pixel(color, c.x0 - x, c.y0 - y);
```

```
        set_color_of_pixel(color, c.x0 + y, c.y0 + x);
        set_color_of_pixel(color, c.x0 - y, c.y0 + x);
        set_color_of_pixel(color, c.x0 + y, c.y0 - x);
        set_color_of_pixel(color, c.x0 - y, c.y0 - x);
        x++;
        if (d < 0)
        {
            d += (4 * x) + 6;
        }
        else
        {
            d += 4 * (x - y) + 10;
            y--;
        }
    }
}

void BMP::fill_circle(Circle c, RGB color)
{
    draw_circle_line(c, color);

    for (int y = -c.r; y <= c.r; y++)
    {
        if (y + c.y0 < 0 || y + c.y0 > info_header.height)
            continue;
        for (int x = -c.r; x <= c.r; x++)
        {
            if (x + c.x0 < 0 || x + c.x0 >= info_header.width)
                continue;
            if (x * x + y * y <= c.r * c.r)
                set_color_of_pixel(color, c.x0 + x, c.y0 + y);
        }
    }
}

std::vector<std::vector<int>> BMP::free_border_data(RGB color)
{
    std::vector<std::vector<int>>
free_border_data_array(info_header.height,
std::vector<int>(info_header.width, 0));
    for (int y = 0; y < info_header.height; y++)
    {
        for (int x = 0; x < info_header.width; x++)
        {
            if (pixels[y][x] == color)
            {
                if ((y + 1 == info_header.height || y - 1 < 0) && (x +
1 == info_header.width || x - 1 < 0))
                    free_border_data_array[y][x] = 5;
                else if (y + 1 == info_header.height || y - 1 < 0 || x
+ 1 == info_header.width || x - 1 < 0)
                    free_border_data_array[y][x] = 3;
                if (y + 1 < info_header.height && pixels[y + 1][x] !=
color)
                    free_border_data_array[y][x] += 1;
                if (y - 1 >= 0 && pixels[y - 1][x] != color)
                    free_border_data_array[y][x] += 1;
```

```
                if (x + 1 < info_header.width && pixels[y][x + 1] !=
color)
                    free_border_data_array[y][x] += 1;
                if (x - 1 >= 0 && pixels[y][x - 1] != color)
                    free_border_data_array[y][x] += 1;
                if (y + 1 < info_header.height && x + 1 <
info_header.width && pixels[y + 1][x + 1] != color)
                    free_border_data_array[y][x] += 1;
                if (y + 1 < info_header.height && x - 1 >= 0 &&
pixels[y + 1][x - 1] != color)
                    free_border_data_array[y][x] += 1;
                if (y - 1 >= 0 && x + 1 < info_header.width &&
pixels[y - 1][x + 1] != color)
                    free_border_data_array[y][x] += 1;
                if (y - 1 >= 0 && x - 1 >= 0 && pixels[y - 1][x - 1]
!= color)
                    free_border_data_array[y][x] += 1;
            }
        }
    }
    return free_border_data_array;
}

void BMP::print_file_header()
{
    using std::hex;
    using std::dec;
    std::cout << "\nFILE HEADER INFO\n";
    std::cout << "signature:     \t0x" << hex << file_header.signature
<< " (" << dec << file_header.signature << ")\n";
    std::cout << "filesize:    \t0x" << hex << file_header.filesize <<
" (" << dec << file_header.filesize << ")\n";
    std::cout << "reserved1:     \t0x" << hex << file_header.reserved1
<< " (" << dec << file_header.reserved1 << ")\n";
    std::cout << "reserved2:     \t0x" << hex << file_header.reserved2
<< " (" << dec << file_header.reserved2 << ")\n";
    std::cout      <<      "pixelArrOffset:\t0x"      <<      hex      <<
file_header.pixelArrOffset      <<      "      ("      <<      dec      <<
file_header.pixelArrOffset << ")\n";
}

void BMP::print_info_header()
{
    using std::hex;
    using std::dec;
    using std::cout;
    cout << "\nINFO HEADER\n";
    cout << "headerSize:    \t0x" << hex << info_header.header_size <<
" (" << dec << info_header.header_size << ")\n";
    cout << "width:          \t0x" << hex << info_header.width << " ("
<< dec << info_header.width << ")\n";
    cout << "height:        \t0x" << hex << info_header.height << " ("
<< dec << info_header.height << ")\n";
    cout << "planes:        \t0x" << hex << info_header.planes << " ("
<< dec << info_header.planes << ")\n";
    cout << "bitsPerPixel: \t0x" << hex << info_header.bitsPerPixel <<
" (" << dec << info_header.bitsPerPixel << ")\n";
```

```
    cout << "compression:   \t0x" << hex << info_header.compression <<
" (" << dec << info_header.compression << ")\n";
    cout << "imageSize:     \t0x" << hex << info_header.image_size << "
(" << dec << info_header.image_size << ")\n";
    cout      <<      "xPixelsPerMeter:\t0x"      <<      hex      <<
info_header.x_pixels_per_meter     <<     "     ("     <<     dec     <<
info_header.x_pixels_per_meter << ")\n";
    cout      <<      "yPixelsPerMeter:\t0x"      <<      hex      <<
info_header.y_pixels_per_meter     <<     "     ("     <<     dec     <<
info_header.y_pixels_per_meter << ")\n";
    cout     <<     "colorsInColorTable:\t0x"     <<     hex     <<
info_header.colors_in_color_table    <<    "    ("    <<    dec    <<
info_header.colors_in_color_table<< ")\n";
    cout     <<     "importantColorCount:\t0x"     <<     hex     <<
info_header.important_color_count    <<    "    ("    <<    dec    <<
info_header.important_color_count << ")\n";
}
```

Bmp.h

```
#pragma once
#include <string>
#include <vector>
#include <string>
#include <iostream>

#pragma pack(push, 1)
struct BitMapFileHeader
{
    uint16_t signature;
    uint32_t filesize;
    uint16_t reserved1;
    uint16_t reserved2;
    uint32_t pixelArrOffset;
};

struct BitmapInfoHeader
{
    uint32_t header_size;
    uint32_t width;
    uint32_t height;
    uint16_t planes;
    uint16_t bitsPerPixel;
    uint32_t compression;
    uint32_t image_size;
    uint32_t x_pixels_per_meter;
    uint32_t y_pixels_per_meter;
    uint32_t colors_in_color_table;
    uint32_t important_color_count;
};

struct RGB
{
    uint8_t b;
    uint8_t g;
    uint8_t r;

    bool operator==(RGB other)
    {
```

```
        return this->b == other.b && this->g == other.g && this->r ==
other.r;
    }
    bool operator!=(RGB other)
    {
        return !(*this==other);
    }
};

struct Coordinates
{
    int x0;
    int y0;
    int x1;
    int y1;
};

struct Circle
{
    int x0;
    int y0;
    int r;
};

#pragma pack(pop)

class BMP
{
public:
    BMP();
    BMP(const char *input_path);

    void read(const char *input_path);
    void write(const char *output_path);
    void print_info_of_pixels();
    void print_file_header();
    void print_info_header();

    void      rgbfilter(std::string      component_name,      uint8_t
component_value);

    void draw_circle(Circle c, RGB color, int thickness);
    void  draw_circle(Circle  c,  RGB  color,  int  thickness,  RGB
fill_color);
    void draw_line(Coordinates coordinates, int thickness, RGB color);
    void draw_circle_line(Circle c, RGB color);

    void split(int number_x, int number_y, int thickness, RGB color);

    void change_pixels_of_field(RGB color, Coordinates coordinates);

    void filled_rects(RGB color, int thickness, RGB border_color);

protected:
    BitmapInfoHeader info_header;
    BitMapFileHeader file_header;
    std::vector<std::vector<RGB>> pixels;
    int padding_size;
```

```
    Coordinates *check_rect(int x0, int y0, RGB color);
    bool check_border_of_rect(Coordinates coordinates, RGB color);
    bool check_line(int x0, int x1, int y, RGB color);
    bool check_border(Coordinates coordinates, RGB color);

    void read_headers(std::ifstream &bmp_file);
    void read_pixels(std::ifstream &bmp_file);

    void write_headers(std::ofstream &out_file);
    void write_pixels(std::ofstream &out_file);
    void set_color_of_pixel(RGB color, int x, int y);
    void    set_component_value_of_pixel(std::string    component_name,
uint8_t component_value, RGB *pixel);
    void fill_circle(Circle c, RGB color);
    std::vector<int> coordinates_split(int num, int len_axis);
    std::vector<std::vector<int>> free_border_data(RGB color);
};
```

Input.cpp

```
#include <iostream>
#include <getopt.h>
#include <cstring>
#include <sstream>
#include "input.h"

using std::cout;
unordered_map<string, string> input(int argc, char **argv)
{
    string last_arg = argv[argc - 1];
    unordered_map<string, string> input;

    const option long_optinons[] = {

        {"filled_rects", no_argument, nullptr, 1},
        {"color", required_argument, nullptr, 1},
        {"border_color", required_argument, nullptr, 1},
        {"thickness", required_argument, nullptr, 1},

        {"circle", no_argument, nullptr, 1},
        {"center", required_argument, nullptr, 1},
        {"radius", required_argument, nullptr, 1},
        {"color", required_argument, nullptr, 1},
        {"thickness", required_argument, nullptr, 1},
        {"fill", no_argument, nullptr, 1},
        {"fill_color", required_argument, nullptr, 1},

        {"rgbfilter", no_argument, nullptr, 1},
        {"component_name", required_argument, nullptr, 1},
        {"component_value", required_argument, nullptr, 1},

        {"split", no_argument, nullptr, 1},
        {"number_x", required_argument, nullptr, 1},
        {"number_y", required_argument, nullptr, 1},
```

```
        {"thickness", required_argument, nullptr, 1},
        {"color", required_argument, nullptr, 1},

        {"input", required_argument, nullptr, 1},

        {"output", required_argument, nullptr, 1},

        {"info", no_argument, nullptr, 1},

        {"help", no_argument, nullptr, 1}};
    int index = -1;
    int rez = 0;
    while ((rez = getopt_long(argc, argv, "i:o:h", long_optinons,
&index)) != -1)
    {
        opterr = 0;
        // if (!rez)
        //  throw_exception("Unknown flag", 42);

        if (rez == 'i')
        {
            string arg(optarg);
            input.insert({"i", arg});
        }

        if (rez == 'o')
        {
            string arg(optarg);
            input.insert({"o", arg});
        }

        if (rez == 'h')
        {
            input.insert({"h", ""})
        }

        if (rez == '?')
        {
            throw_exception("Unknown flag", 43);
        }

        if (long_optinons[index].has_arg == no_argument)
        {
            input.insert({long_optinons[index].name, ""});
        }

        if (long_optinons[index].has_arg == required_argument)
        {
            string arg(optarg);
            input.insert({long_optinons[index].name, arg});
        }
    }

    if (input.find("input") == input.end() && input.find("i") ==
input.end())
    {
        for (auto v : input)
        {
```

```
            if (v.second == last_arg)
                throw_exception("Enter input path of file", 44);
        }
        input.insert({"input", last_arg});
    }
    return input;
}

void throw_exception(string message, int exit_code)
{
    std::cerr << message << '\n';
    exit(exit_code);
}


void print_map(unordered_map<string, string> dict)
{
    for (const auto v : dict)
    {
        cout << v.first << " " << v.second << "\n";
    }
}

input_data *check_input(unordered_map<string, string> input)
{
    vector<vector<string>> samples = {
        {"filled_rects", "color", "border_color", "thickness"},
        {"circle", "center", "radius", "thickness", "color", "fill",
"fill_color"},
    {"rgbfilter", "component_name", "component_value"},
    {"split", "number_x", "number_y", "color", "thickness"}};
input_data *tmp;
int is_input_data = 1;
tmp = new input_data;
string input_path;
string output_path = "output.bmp";
if (input.find("input") != input.end())
{
    input_path = input["input"];
}
else if (input.find("i") != input.end())
{
    input_path = input["i"];
}

if (input.find("output") != input.end())
{
    output_path = input["output"];
}
else if (input.find("o") != input.end())
{
    output_path = input["o"];
}
if (input_path == output_path)
    throw_exception("input path and output path are same", 44);

for (auto v : samples)
{
```

```
        is_input_data = 1;
        for (auto w : v)
        {
            if (input.find(w) == input.end() && w != "fill_color" && w
!= "fill")
            {
                is_input_data = -1;
                delete tmp;
                // cout << w << '\n';
                tmp = new input_data;
                break;
            }
            if (w == "color")
            {
                vector<int>     color     =     parse_input(input[w],
check_rgb_value);
                if (color.size() != 3)
                    throw_exception(input[w] + " not requaries to the
template of flag "  + w, 42);
                tmp->color.r = color[0];
                tmp->color.g = color[1];
                tmp->color.b = color[2];
            }
            else if (w == "border_color")
            {
                vector<int>   border_color   =   parse_input(input[w],
check_rgb_value);
                if (border_color.size() != 3)
                    throw_exception(input[w] + " not requaries to the
template \"X.X.X\" of flag " + w, 42);
                tmp->border_color.r = border_color[0];
                tmp->border_color.g = border_color[1];
                tmp->border_color.b = border_color[2];
            }
            else if (w == "thickness")
            {
                tmp->thickness = custom_stoi(input[w]);
                check_positive(tmp->thickness);
            }
            else if (w == "center")
            {
                vector<int> coordinates = parse_input(input[w]);
                if (coordinates.size() != 2)
                    throw_exception(input[w] + " not requaries to the
template \"X.X\" of flag " + w, 42);
                tmp->circle.x0 = coordinates[0];
                tmp->circle.y0 = coordinates[1];
            }
            else if (w == "radius")
            {
                tmp->circle.r = custom_stoi(input[w]);
                check_positive(tmp->circle.r);
            }
            else if (w == "fill_color" && input.find(w) != input.end())
            {
                vector<int>    fill_color    =    parse_input(input[w],
check_rgb_value);
                if (fill_color.size() != 3)
```

```
                    throw_exception(input[w] + " not requaries to the
template \"X.X.X\" of flag " + w, 42);
                tmp->fill_color.r = fill_color[0];
                tmp->fill_color.g = fill_color[1];
                tmp->fill_color.b = fill_color[2];
            }
            else if (w == "component_name")
            {
                check_component_name(input[w]);
                tmp->component_name = input[w];
            }
            else if (w == "component_value")
            {
                tmp->component_value = custom_stoi(input[w]);
                check_rgb_value(tmp->component_value);
            }
            else if (w == "number_x")
            {
                tmp->number_x = custom_stoi(input[w]);
                check_positive(tmp->number_x + 1);
            }
            else if (w == "number_y")
            {
                tmp->number_y = custom_stoi(input[w]);
                check_positive(tmp->number_y + 1);
            }
            else if (w == "fill" && input.find(w)!=input.end())
            {
                // cout << w;
                tmp->fill = true;
            }
        }
        if(is_input_data == 1)
            break;
    }
    string function;
    for (size_t i = 0; i<samples.size(); i++)
    {
        if(input.find(samples[i][0]) != input.end())
            function = samples[i][0];
    }
    if (is_input_data == -1)
    {
        throw_exception("the " + function + " function accepts other
flags", 44);
    }
    tmp->function = function;
    tmp->input_path = input_path;
    tmp->output_path = output_path;
    return tmp;
}
vector<int> parse_input(string str, void(*func)(int))
{
    vector<int> output;
    vector<string> array = split(str, '.');
    for (size_t i = 0; i < array.size(); i++)
    {
        int t = custom_stoi(array[i]);
```

```
        func(t);
        output.push_back(t);
    }
    return output;
}

vector<int> parse_input(string str)
{
    vector<int> output;
    vector<string> array = split(str, '.');
    for (size_t i = 0; i < array.size(); i++)
    {
        int t = custom_stoi(array[i]);
        output.push_back(t);
    }
    return output;
}

void check_rgb_value(int val)
{
    if (val < 0 || val > 255)
        throw_exception("The value of color must be in range [0, 255]",
41);
}

vector<string> split(string str, char ch)
{
    std::istringstream ss(str);
    string token;
    vector<string> array;
    while (getline(ss, token, ch))
        array.push_back(token);
    return array;
}

int custom_stoi(string str)
{
    int n;
    try
    {
        n = std::stoi(str);
    }
    catch (const std::exception &e)
    {
        throw_exception("The value must be integer", 41);
    }
    return n;
}

void check_positive(int val)
{
    if (val < 0)
        throw_exception("The value must be >= 0", 40);
}

void check_component_name(string str)
{
    if (str != "red" && str != "blue" && str != "green")
```

```
        throw_exception("Value must be red, green or blue", 40);
}

void print_help()
{
    cout << "Course work for option 5.8, created by Kirill Verdin." <<
'\n\n';
    cout << "Options:\n\n"
    "--input/-i - enter path of input file\n"
    "--output/-o - enter path of output file\n\n"
    "--info - print info about image\n\n"
    "--help/-h - print list of functions\n\n"
    "--filled_rects - find all rectangles of the certain color\n"
    "--color <r.g.b> - color of rectangles\n"
    "--border_color <r.g.b> - border_color of rectangles\n"
    "--thickness - thickness of border line of rectange\n\n"
    "--circle - draw circle\n"
    "--center <x.y> - coordinates of the center of circle\n"
    "--radius - radius of circle\n"
    "--color <r.g.b> - color of border of circle\n"
    "--thickness - thickness of border line\n"
    "--fill - fill/not fill circle\n"
    "--fill_color <r.g.b> color of circle\n\n"
    "--rgbfilter - change the value of 1 component of color for whole
image\n"
    "--component_name <red/green/blue> - name of RGB component that
would change\n"
    "--component_value -- value of RGB component\n\n"
    "--split - split image on x*y parts\n"
    "--number_x - number of parts on axis y\n"
    "--number_y - number of parts on axis x\n"
    "-thickness - thickness of line\n"
    "--color <r.g.b> - color of line\n";
}
```

Input.h

```
#pragma once

#include "../bmp/bmp.h"
#include <string>
#include <unordered_map>
#include <vector>

using std::string;
using std::unordered_map;
using std::vector;

struct input_data
{
    string function = "";
    string input_path;
    string output_path = "output.bmp";
    RGB color;
    RGB border_color;
    RGB fill_color;
    int thickness = 0;
```

```
    Circle circle;
    bool fill = false;
    string component_name = "";
    int component_value = -1;
    int number_x = -1;
    int number_y = -1;
};


void throw_exception(string message, int exit_code);
void print_map(unordered_map<string, string>);
unordered_map<string, string> input(int argc, char **argv);
vector<int> parse_input(string str, void(*func)(int));
vector<int> parse_input(string str);
void check_rgb_value(int value);
vector<string> split(string str, char ch);
int custom_stoi(string str);
void check_positive(int val);

input_data *check_input(unordered_map<string, string> input);
void check_component_name(string str);
void print_help();
```

main.cpp

```
#include "input/input.h"
#include "bmp/bmp.h"
#include <iostream>

int main(int argc, char** argv)
{
    unordered_map<string, string> input_info = input(argc, argv);
    if     (input_info.find("help")     !=     input_info.end()     ||
input_info.find("h") != input_info.end())
    {
        print_help();
        if (input_info.size() == 2)
            return 0;
    }
    input_data* data = check_input(input_info);
    BMP image;
    image.read(data->input_path.c_str());
    if(input_info.find("info") != input_info.end())
    {
        image.print_file_header();
        image.print_info_header();
    }
    if (data->function == "filled_rects")
    {
        image.filled_rects(data->color,     data->thickness,     data-
>border_color);
    }
    else if (data->function == "circle")
    {
        if (data->fill)
```

```
            image.draw_circle(data->circle,     data->color,     data-
>thickness, data->fill_color);

        else
            image.draw_circle(data->circle,     data->color,     data-
>thickness);
    }
    else if (data->function == "rgbfilter")
    {
        image.rgbfilter(data->component_name, data->component_value);
    }
    else if (data->function == "split")
    {
        image.split(data->number_x,  data->number_y,  data->thickness,
data->color);
    }
    image.write(data->output_path.c_str());
    return 0;
}
```