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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка изображений

Шаповаленко Е.В.

Студент гр. 3341 _____

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2024

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Шаповаленко Е. В.

Группа 3341

Тема работы: Обработка изображения

Вариант 5.8

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

Программа должна реализовывать весь следующий функционал по обработке bmp-файла

Общие сведения

24 бита на цвет, без сжатия.

Файл может не соответствовать формату BMP, т.е. необходимо проверка на BMP формат (дополнительно стоит помнить, что версий у формата несколько). Если файл не соответствует формату BMP или его версии, то программа должна завершиться с соответствующей ошибкой.

Обратите внимание на выравнивание; мусорные данные, если их необходимо дописать в файл для выравнивания, должны быть нулями.

Обратите внимание на порядок записи пикселей.

Все поля стандартных BMP заголовков в выходном файле должны иметь те же значения что и во входном (

разумеется, кроме тех, которые должны быть изменены).

Программа должна иметь следующую функции по обработке изображений:

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

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`)

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

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)

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

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 27.05.2024

Дата защиты реферата: 29.05.2024

Студент _____ Шаповаленко Е.В.

Преподаватель _____ Глазунов С.А.

_____

АННОТАЦИЯ

Курсовой проект по варианту 5.8 представляет собой программу, которая обрабатывает изображение формата BMP, введенное пользователем. Программа использует стандартные потоки ввода и вывода. Программа предлагает пользователю выбор из нескольких функций для обработки изображения, включая рисование рамки для всех прямоугольников указанного цвета, рисование окружности, изменение указанной компоненты пикселей и разделение изображения на указанное количество частей. Все эти функции реализованы с использованием функций стандартной библиотеки. Каждая подзадача вынесена в отдельную функцию, а функции сгруппированы в несколько файлов. Также написан Makefile для компиляции программы. Программа завершает работу после выполнения одного из действий, выбранных пользователем.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель данной работы - разработать программу для обработки изображения, введенного пользователем, с использованием стандартных потоков ввода и вывода и Makefile для компиляции.

Для достижения поставленной цели требуется решить следующие задачи:

1. Изучение структуры BMP файла
2. Изучение структуры реализации CLI (Command Line Interface)
3. Изучение стандартных библиотек, которые будут использоваться в программе
4. Написание функций, включая рисование рамки для всех прямоугольников указанного цвета, рисование окружности, изменение указанной компоненты пикселей и разделение изображения на указанное количество частей.
5. Сборка программы с использованием Makefile
6. Тестирование программы на различных входных данных

Исходный код программы см. в приложении А.

Тестирование работы программы см. в приложении В.

1. ХОД ВЫПОЛНЕНИЯ РАБОТЫ

1.1. Структуры данных

Структуры, необходимые для работы программы описаны в файле: structures.h

Struct Coord используется для описания координаты. Два поля x и y хранят координату по “x” и “y” соответственно.

struct Color используется для описания цвета. Три поля r, g и b хранят красную, зеленую и синюю компоненту цвета соответственно. В структуре переопределены некоторые из операторов для удобства пользования.

struct BMPHeader используется для хранения заголовка BMP из изображения. Структура имеет поля, аналогичные таковым в заголовке BMP внутри файла с изображением.

struct DIBHeader используется для хранения заголовка DIB из изображения. Структура имеет поля, аналогичные таковым в заголовке DIB внутри файла с изображением.

struct RGB используется для хранения информации о пикселях изображения. Структура аналогична по строению структуре Color, однако использует другой тип данных для хранения цвета пикселя. Это сделано для удобства считывания информации из файла.

У структур BMPHeader, DIBHeader и RGB убрано выравнивание для того, чтобы корректно считать данные из файла.

1.2. Ввод/вывод

Обработка ввода пользователя осуществляется классом Handler, описанном в файлах handler.h и handler.cpp. Метод getFlags() получает флаги, введенные пользователем при помощи функций из библиотеки getopt.h, и если был введен неправильный флаг, то завершает работу программы с сообщением об ошибке. Метод getFinFoutNames() считывает названия входного и выходного файлов в соответствии с введенными/невведенными флагами “--input” (“-i”) и “--output” (“-o”). По умолчанию имя выходного файла “out.bmp”.

Для всех других ошибок также предусмотрен вывод информации о том, какая ошибка возникла, после которого программа завершает работу.

Вывод информации о программе осуществляется методом doHelp() класса Handler. Функция выводит набор инструкций по работе с программой.

Вывод информации об изображении осуществляется методом showImageInfo() класса ImageBMP. Функция выводит содержимое заголовков BMP файла на экран.

Считывание изображения осуществляется методом readImageFromFile() класса ImageBMP. Данные из файла записываются в структуры BMPHeader, DIBHeader и RGB функцией fread(). Выделение памяти для хранения пикселей осуществляется методом allocateMemmoryForPixels() класса ImageBMP, учитывающим выравнивание памяти в BMP файле.

Запись изображения осуществляется методом writeImageToFile() класса ImageBMP. Данные из структур BMPHeader, DIBHeader и RGB записываются в файл функцией fwrite(). Очистка памяти для хранения пикселей осуществляется методом freeMemmoryForPixels() класса ImageBMP.

1.3. Проверка соблюдения формата ввода

Методы класса Handler вида isFunc(), где Func() это некоторая функция по обработке изображения, проверяют введенные флаги на соответствие требованиям функции. Это делается проверкой, что все необходимые флаги введены, а все лишние флаги — нет. Это осуществляется методами getRedundantFlags() и checkFlagCompliance(). Если набор флагов не подходит ни под одну из функций, программа завершает работу с ошибкой. Иначе выполняется соответствующий метод doFunc() (например, isHelp() и doHelp()).

Внутри методов класса Handler вида doFunc(), где Func() это некоторая функция по обработке изображения, вызываются функции с префиксом “parse”. Эти функции проверяют соблюдение формата введенных пользователем данных, а также то, что значения этих данных находятся в допустимых границах. Если эти условия не выполняются, программа завершает работу с ошибкой.

Метод readImageFromFile() класса ImageBMP после считывания заголовков производит проверку формата изображения функцией checkFormat(). Если сигнатура файла не совпадает с форматом BMP (“424d” или “4d42”), если у файла есть сжатие или если на один пиксель отведено не 24 бит, то программа завершает работу с соответствующей ошибкой.

1.4. Обработка изображения

Считывание, хранение, обработка и запись изображения осуществляется классом ImageBMP. Считывание флагов и вызов выбранной пользователем функции осуществляется классом Handler. Внутри методов класса Handler вида doFunc() создается объект класса ImageBMP, у которого вызываются методы, необходимые для выполнения требуемой обработки изображения.

Метод checkCoordsValidity() класса Handler проверяет, находятся ли полученные на вход координаты внутри изображения или нет.

Методы getWidth() и getHeight() класса Handler возвращают ширину и высоту изображения соответственно.

Метод clearPixels() класса Handler очищает все пиксели (задает всем пикселям черный цвет (“0.0.0”)).

Методы getColor() и setColor() класса Handler возвращают цвет пикселя и присваивают цвет пикселю соответственно.

Метод setSize() класса Handler задает изображению новые ширину и высоту, с очисткой пикселей.

Метод copy() класса Handler возвращает область, описанную координатами верхнего левого и правого нижнего углов, в виде объекта класса ImageBMP.

Метод doBorderRectangles() класса Handler обводит все прямоугольники заданного пользователем цвета рамкой, которой можно задать цвет и толщину линий. Для этого вызывается метод borderRectangles() класса ImageBMP. Метод осуществляет поиск прямоугольников (координат левого верхнего и правого нижнего угла). По координатам найденного прямоугольника рисуется рамка методом drawLine() класса ImageBMP.

Метод doCircle() класса Handler рисует окружность во заданным пользователем координатам центра и радиусу, у которой можно задать цвет и толщину границы и цвет заливки. Для этого вызывается метод drawCircle() класса ImageBMP. Метод закрашивает пиксели, которые удовлетворяют уравнению окружности, что проверяется методами checkOnCircle() и checkInCircle(). После этого рисуется две окружности drawBresenhamCircle() класса ImageBMP для сглаживания неровностей.

Метод doRGBFilter() класса Handler присваивает введенное пользователем значение указанной пользователем компоненте каждого пикселя. Для этого вызывается метод rgbFiler() класса ImageBMP.

Метод doSplit() класса Handler разделяет изображение на указанное пользователем количество частей по вертикали и горизонтали линиями, у которых задается цвет и толщина. Для этого высчитываются координаты начала и конца каждой из линий, после чего они рисуются методом drawLine() класса ImageBMP.

В функции main() в файле main.cpp создается объект класса Handler, который и производит обработку запроса пользователя.

1.5. Makefile

Makefile описывает процесс компиляции программы с именем “cw”. Он использует компилятор G++.

Цель “all” компилирует программу “cw” из объектных файлов и динамической библиотекой “libimageBMP.so”.

Цель “libimageBMP.so” компилирует одноименную динамическую библиотеку по обработке изображения.

Цель “clear” удаляет исполняемый файл “cw”, динамическую библиотеку “libimageBMP.so” и все объектные файлы.

ЗАКЛЮЧЕНИЕ

В результате выполнения данной работы были решены следующие задачи:

1. Изучена структура BMP файла.
2. Изучена структура реализации CLI (Command Line Interface).
3. Изучены стандартные библиотеки, которые использовались в программе.
4. Разработаны функции, которые позволяют обрабатывать изображение, введенное пользователем. Они включают рисование рамки для всех прямоугольников указанного цвета, рисование окружности, изменение указанной компоненты пикселей и разделение изображения на указанное количество частей.
5. Для сборки программы был использован файл Makefile, который автоматизирует процесс компиляции и линковки программы. Он содержит необходимые команды для сборки программы и её зависимостей.
6. Программа была протестирована на различных входных данных, чтобы проверить её корректность и работоспособность. Тестирование позволило убедиться, что программа выполняет поставленные задачи правильно и обрабатывает изображение в соответствии с требованиями.

В итоге, разработанная программа успешно обрабатывает изображение, введенное пользователем в соответствии с условием задачи, с использованием стандартных потоков ввода и вывода.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include "handler.h"

int main(int argc, char **argv)
{
    hdlr::Handler handler;

    handler.show_author_info("5.8", "Egor", "Shapovalenko");
    handler.getFlags(argc, argv);
    handler.handleFlags();

    return 0;
}
```

Название файла: handler.h

```
#ifndef HANDLER
#define HANDLER

#include <getopt.h>
#include <map>
#include <string>
#include <set>

#define HANDLER_ERROR           45

#define FLAGS_NUMBER            23

#define HELP_IDX                'h'
#define INPUT_IDX               'i'
#define OUTPUT_IDX              'o'
#define INFO_IDX                1001

#define THICKNESS_IDX           1002
#define COLOR_IDX               1003
#define FILL_IDX                1004
#define FILL_COLOR_IDX          1005

#define BORDER_RECTS_IDX        1006
#define BORDER_COLOR_IDX        1007

#define CIRCLE_IDX              1008
#define CENTER_IDX              1009
#define RADIUS_IDX              1010

#define RGBFILTER_IDX           1011
#define COMPONENT_NAME_IDX      1012
#define COMPONENT_VALUE_IDX     1013

#define SPLIT_IDX               1014
#define NUMBER_X_IDX            1015
```

```
#define NUMBER_Y_IDX            1016

#define CONCAT_IDX              1017
#define AXIS_IDX                1018
#define INPUT_SECOND_IDX        1019

namespace hdlr
{

void throwError(const char *message, int exit_code);

struct Flag
{
    bool entered;
    std::string parameter;
};

class Handler
{
private:
    std::map<int, Flag>    flags;
    const char             *short_options;
    const struct option    long_options[FLAGS_NUMBER];
    std::string            last_argument;

    std::set<int> getRedundantFlags(std::set<int>& required_flags,
        std::set<int>& optional_flags);

    bool checkFlagCompliance(std::set<int>& required_flags,
        std::set<int>& redundant_flags);

    void   getFinFoutNames(std::string   &input_file_name,   std::string
&output_file_name);
    void getSecondFinName(std::string &second_file_name);

    bool isHelp();
    bool isInfo();
    bool isBorderRectangles();
    bool isCircle();
    bool isRGBFilter();
    bool isSplit();
    bool isConcat();

    void doHelp();
    void doInfo();
    void doBorderRectangles();
    void doCircle();
    void doRGBFilter();
    void doSplit();
    void doConcat();

public:
    Handler();
    ~Handler();
    void show_author_info(const char *option, const char *name, const char
*surname);
    void getFlags(int argc, char **argv);
```

```
    void handleFlags();
};
}

#endif
```

Название файла: handler.cpp

```
#include "handler.h"
#include "parser.h"

#include <getopt.h>
#include <stdio.h>
#include <map>
#include <string>
#include <set>
#include <algorithm>


hdlr::Handler::Handler() :
    flags{
        {HELP_IDX,              {0, ""}},
        {INPUT_IDX,             {0, ""}},
        {OUTPUT_IDX,            {0, ""}},
        {INFO_IDX,              {0, ""}},

        {THICKNESS_IDX,         {0, ""}},
        {COLOR_IDX,             {0, ""}},
        {FILL_IDX,              {0, ""}},
        {FILL_COLOR_IDX,        {0, ""}},

        {BORDER_RECTS_IDX,      {0, ""}},
        {BORDER_COLOR_IDX,      {0, ""}},

        {CIRCLE_IDX,            {0, ""}},
        {CENTER_IDX,            {0, ""}},
        {RADIUS_IDX,            {0, ""}},

        {RGBFILTER_IDX,         {0, ""}},
        {COMPONENT_NAME_IDX,    {0, ""}},
        {COMPONENT_VALUE_IDX,   {0, ""}},

        {SPLIT_IDX,             {0, ""}},
        {NUMBER_X_IDX,          {0, ""}},
        {NUMBER_Y_IDX,          {0, ""}},

        {CONCAT_IDX,            {0, ""}},
        {AXIS_IDX,              {0, ""}},
        {INPUT_SECOND_IDX,      {0, ""}},
    },
    short_options("hi:o:"),
    long_options{
        {"help",            no_argument,          NULL,    HELP_IDX},
        {"input",           required_argument,    NULL,    INPUT_IDX},
        {"output",          required_argument,    NULL,    OUTPUT_IDX},
        {"info",            no_argument,          NULL,    INFO_IDX},
```

```
        {"thickness",       required_argument,    NULL,    THICKNESS_IDX},
        {"color",           required_argument,    NULL,    COLOR_IDX},
        {"fill",            no_argument,          NULL,    FILL_IDX},
        {"fill_color",      required_argument,    NULL,    FILL_COLOR_IDX},

        {"filled_rects",    no_argument,        NULL,   BORDER_RECTS_IDX},
        {"border_color",            required_argument,              NULL,
BORDER_COLOR_IDX},

        {"circle",          no_argument,          NULL,    CIRCLE_IDX},
        {"center",          required_argument,    NULL,    CENTER_IDX},
        {"radius",          required_argument,    NULL,    RADIUS_IDX},

        {"rgbfilter",       no_argument,          NULL,    RGBFILTER_IDX},
        {"component_name",        required_argument,                NULL,
COMPONENT_NAME_IDX},
        {"component_value",     required_argument,                  NULL,
COMPONENT_VALUE_IDX},

        {"split",           no_argument,          NULL,    SPLIT_IDX},
        {"number_x",        required_argument,    NULL,    NUMBER_X_IDX},
        {"number_y",        required_argument,    NULL,    NUMBER_Y_IDX},

        {"concat",          no_argument,          NULL,    CONCAT_IDX},
        {"axis",            required_argument,    NULL,    AXIS_IDX},
        {"input_second",            required_argument,              NULL,
INPUT_SECOND_IDX},

        {NULL,              0,                    NULL,    0}
    }
{}

hdlr::Handler::~Handler() = default;

void hdlr::Handler::show_author_info(const char *option, const char *name,
const char *surname)
{
    printf("Course work for option %s, created by %s %s\n", option, name,
surname);
}

void hdlr::throwError(const char *message, int exit_code)
{
    printf("%s\n", message);
    exit(exit_code);
}

void hdlr::Handler::getFlags(int argc, char **argv)
{
    opterr = 0;
    int option;
    while ((option = getopt_long(argc, argv, short_options, long_options,
NULL)) != -1) {
        if (flags.find(option) != flags.end()) {
            flags[option].entered = true;
            if (optarg) {
                flags[option].parameter = optarg;
```

```
            }
        } else {
            throwError("Error: wrong flag.", HANDLER_ERROR);
        }
    }
    last_argument = argv[argc-1];
}

std::set<int>            hdlr::Handler::getRedundantFlags(std::set<int>&
required_flags,
    std::set<int>& optional_flags)
{
    std::set<int> redundant_flags;

    for (auto i : flags) {
        if (required_flags.find(i.first) == required_flags.end() &&
            optional_flags.find(i.first) == optional_flags.end()) {
            redundant_flags.insert(i.first);
        }
    }

    return redundant_flags;
}

bool  hdlr::Handler::checkFlagCompliance(std::set<int>&  required_flags,
std::set<int>& redundant_flags) {
    bool all_required_entered = true;
    for (auto i : required_flags) {
        all_required_entered &= flags[i].entered;
    }

    bool all_redundant_not_entered = true;
    for (auto i : redundant_flags) {
        all_redundant_not_entered &= !flags[i].entered;
    }
    return all_required_entered && all_redundant_not_entered;
}

void     hdlr::Handler::getFinFoutNames(std::string     &input_file_name,
std::string &output_file_name)
{
    if (flags[INPUT_IDX].entered) {
        input_file_name = flags[INPUT_IDX].parameter;
    } else {
        input_file_name = last_argument;
    }

    if (flags[OUTPUT_IDX].entered) {
        output_file_name = flags[OUTPUT_IDX].parameter;
    }
}

void hdlr::Handler::getSecondFinName(std::string &second_file_name)
{
    if (flags[INPUT_SECOND_IDX].entered) {
        second_file_name = flags[INPUT_SECOND_IDX].parameter;
    }
```

```
}

bool hdlr::Handler::isHelp()
{
    std::set<int> required_flags = {};
    std::set<int> optional_flags = {HELP_IDX};
    std::set<int>  redundant_flags  =  getRedundantFlags(required_flags,
optional_flags);
    return checkFlagCompliance(required_flags, redundant_flags);
}

bool hdlr::Handler::isInfo()
{
    std::set<int> required_flags = {INFO_IDX};
    std::set<int> optional_flags = {INPUT_IDX};
    std::set<int>  redundant_flags  =  getRedundantFlags(required_flags,
optional_flags);
    return checkFlagCompliance(required_flags, redundant_flags);
}

bool hdlr::Handler::isBorderRectangles()
{
    std::set<int>   required_flags   =   {BORDER_RECTS_IDX,   COLOR_IDX,
BORDER_COLOR_IDX, THICKNESS_IDX};
    std::set<int> optional_flags = {INPUT_IDX, OUTPUT_IDX};
    std::set<int>  redundant_flags  =  getRedundantFlags(required_flags,
optional_flags);
    return checkFlagCompliance(required_flags, redundant_flags);
}

bool hdlr::Handler::isCircle()
{
    std::set<int> required_flags = {CIRCLE_IDX, CENTER_IDX, RADIUS_IDX,
THICKNESS_IDX, COLOR_IDX};
    std::set<int>  optional_flags  =  {INPUT_IDX,  OUTPUT_IDX,  FILL_IDX,
FILL_COLOR_IDX};
    std::set<int>  redundant_flags  =  getRedundantFlags(required_flags,
optional_flags);
    return      checkFlagCompliance(required_flags,      redundant_flags)
&& !(flags[FILL_IDX].entered && !flags[FILL_COLOR_IDX].entered);
}

bool hdlr::Handler::isRGBFilter()
{
    std::set<int>  required_flags  =  {RGBFILTER_IDX,  COMPONENT_NAME_IDX,
COMPONENT_VALUE_IDX};
    std::set<int> optional_flags = {INPUT_IDX, OUTPUT_IDX};
    std::set<int>  redundant_flags  =  getRedundantFlags(required_flags,
optional_flags);
    return checkFlagCompliance(required_flags, redundant_flags);
}

bool hdlr::Handler::isSplit()
{
    std::set<int> required_flags = {SPLIT_IDX, NUMBER_X_IDX, NUMBER_Y_IDX,
THICKNESS_IDX, COLOR_IDX};
    std::set<int> optional_flags = {INPUT_IDX, OUTPUT_IDX};
```

```
    std::set<int>  redundant_flags  =  getRedundantFlags(required_flags,
optional_flags);
    return checkFlagCompliance(required_flags, redundant_flags);
}

bool hdlr::Handler::isConcat()
{
    std::set<int>     required_flags     =     {CONCAT_IDX,     AXIS_IDX,
INPUT_SECOND_IDX, COLOR_IDX};
    std::set<int> optional_flags = {INPUT_IDX, OUTPUT_IDX};
    std::set<int>  redundant_flags  =  getRedundantFlags(required_flags,
optional_flags);
    return checkFlagCompliance(required_flags, redundant_flags);
}

void hdlr::Handler::doHelp()
{
    printf("--help (-h)            Prints the help\n");
    printf("--input (-i)            Sets the name of the input image. If
the flag is omitted, it is assumed that the name of the input image is
passed as the last argument\n");
    printf("--output (-o)          Sets the name of the output image. If
the flag is omitted, it is assumed that the name of the input image is
out.bmp\n");
    printf("--info                Prints information about the image\n");
    printf("\n");

    printf("Basic flags:\n");
    printf("--thickness            Line thickness. Format: NUMBER\n");
    printf("--color               Line color. Format: RED.GREEN.BLUE\n");
    printf("--fill                  Works as a binary value: there is a
flag - true, there is no flag - false\n");
    printf("--fill_color           Fill color. Can be used without --fill
flag. Format: RED.GREEN.BLUE\n");
    printf("\n");

    printf("--filled_rects          Drawing a border for all rectangles
filled with specified color. Required: --color, --border_color, --
thickness\n");
    printf("--color               Color that rectangles are filled with.
Format: RED.GREEN.BLUE\n");
    printf("--border_color                Color of the border. Format:
RED.GREEN.BLUE\n");
    printf("\n");

    printf("--circle               Drawing a circle. Required: --center,
--radius, --thickness, --color. Optional: --fill, --fill_color\n");
    printf("--center                 Coordinates of the center. Format:
x.y\n");
    printf("--radius               Radius. Format: NUMBER\n");
    printf("\n");

    printf("--rgbfilter            RGB filter component\n");
    printf("--component_name        Which component needs to be changed.
Format: {red || green || blue}\n");
    printf("--component_value      To which value it needs to be changed.
Format: NUMBER\n");
```

```
    printf("\n");

    printf("--split              Divides the image into number_y*number_x
parts. Required: --thickness, --color\n");
    printf("--number_x             The number of parts along the Y-axis.
Format: NUMBER\n");
    printf("--number_y             The number of parts along the X-axis.
Format: NUMBER\n");
    printf("\n");

    printf("--concat                  Concatenates to given images with
alternating pixels along specified axis. Empty spaces are filled with
specified color. Required: --axis, --input_second, --color\n");
    printf("--input_second       Sets the name of the image to concatenate
with.\n");
    printf("--axis               Along which axis to concatenate. Format:
{x || y}\n");
    printf("\n");
}

void hdlr::Handler::doInfo()
{
    std::string input_file_name;

    if (flags[INPUT_IDX].entered) {
        input_file_name = flags[INPUT_IDX].parameter;
    } else {
        input_file_name = last_argument;
    }

    ie::ImageBMP image_bmp;
    image_bmp.readImageFromFile(input_file_name.c_str());
    image_bmp.showImageInfo();
}

void hdlr::Handler::doBorderRectangles()
{
    int thickness;
    ie::Color rectangles_color, border_color;
    std::string input_file_name;
    std::string output_file_name = "out.bmp";

    psr::parseNumber(thickness, flags[THICKNESS_IDX].parameter);
    psr::parseColor(rectangles_color, flags[COLOR_IDX].parameter);
    psr::parseColor(border_color, flags[BORDER_COLOR_IDX].parameter);
    getFinFoutNames(input_file_name, output_file_name);

    psr::checkValueValidity(thickness,   [](int   thickness)   {   return
(thickness > 0); });

    ie::ImageBMP image_bmp;
    image_bmp.readImageFromFile(input_file_name.c_str());
    image_bmp.borderRectangles(rectangles_color,            border_color,
thickness);
    image_bmp.writeImageToFile(output_file_name.c_str());
}
```

```
void hdlr::Handler::doCircle()
{
    int x0, y0, radius, thickness;
    bool fill;
    ie::Color color, fill_color;
    std::string input_file_name;
    std::string output_file_name = "out.bmp";

    psr::parseCoords(x0, y0, flags[CENTER_IDX].parameter);
    psr::parseNumber(radius, flags[RADIUS_IDX].parameter);
    psr::parseNumber(thickness, flags[THICKNESS_IDX].parameter);
    psr::parseColor(color, flags[COLOR_IDX].parameter);
    fill = flags[FILL_IDX].entered;
    if (fill) {
        psr::parseColor(fill_color, flags[FILL_COLOR_IDX].parameter);
    }
    getFinFoutNames(input_file_name, output_file_name);

    psr::checkValueValidity(radius, [](int radius) { return (radius >
0); });
    psr::checkValueValidity(thickness, [](int thickness) { return
(thickness > 0); });

    ie::ImageBMP image_bmp;
    image_bmp.readImageFromFile(input_file_name.c_str());
    image_bmp.drawCircle(x0, y0, radius, thickness, color, fill,
fill_color);
    image_bmp.writeImageToFile(output_file_name.c_str());
}

void hdlr::Handler::doRGBFilter()
{
    int component_idx, component_value;
    std::string input_file_name;
    std::string output_file_name = "out.bmp";

    psr::parseComponentName(component_idx,
flags[COMPONENT_NAME_IDX].parameter);
    psr::parseNumber(component_value,
flags[COMPONENT_VALUE_IDX].parameter);
    getFinFoutNames(input_file_name, output_file_name);

    psr::checkValueValidity(component_value, [](int component_value)
{ return (component_value >= 0 && component_value <= 255); });

    ie::ImageBMP image_bmp;
    image_bmp.readImageFromFile(input_file_name.c_str());
    image_bmp.rgbFilter(component_idx, component_value);
    image_bmp.writeImageToFile(output_file_name.c_str());
}

void hdlr::Handler::doSplit()
{
    int number_x, number_y, thickness;
    ie::Color color;
    std::string input_file_name;
    std::string output_file_name = "out.bmp";
```

```
    psr::parseNumber(number_x, flags[NUMBER_X_IDX].parameter);
    psr::parseNumber(number_y, flags[NUMBER_Y_IDX].parameter);
    psr::parseNumber(thickness, flags[THICKNESS_IDX].parameter);
    psr::parseColor(color, flags[COLOR_IDX].parameter);
    getFinFoutNames(input_file_name, output_file_name);

    psr::checkValueValidity(number_x,    [](int    number_x)    {    return
(number_x > 1); });
    psr::checkValueValidity(number_y,    [](int    number_y)    {    return
(number_y > 1); });
    psr::checkValueValidity(thickness,   [](int   thickness)   {   return
(thickness > 0); });

    ie::ImageBMP image_bmp;
    image_bmp.readImageFromFile(input_file_name.c_str());

    for (int i = 1; i < number_y; i++) {
        int y = (image_bmp.getHeight() / number_y) * i;
        for (int j = 0; j <= thickness/2; j++) {
            image_bmp.drawLine(0,  y-j,  image_bmp.getWidth()-1,  y-j,  1,
color);
            image_bmp.drawLine(0,  y+j,  image_bmp.getWidth()-1,  y+j,  1,
color);
        }
    }

    For (int i = 1; i < number_x; i++) {
        int x = (image_bmp.getWidth() / number_x) * i;
        for (int j = 0; j <= thickness/2; j++) {
            image_bmp.drawLine(x-j,  0,  x-j,  image_bmp.getHeight()-1,  1,
color);
            image_bmp.drawLine(x+j,  0,  x+j,  image_bmp.getHeight()-1,  1,
color);
        }
    }

    image_bmp.writeImageToFile(output_file_name.c_str());
}

void hdlr::Handler::doConcat()
{
    int axis_idx;
    ie::Color fill_color;
    std::string input_file_name;
    std::string second_file_name;
    std::string output_file_name = "out.bmp";

    psr::parseAxis(axis_idx, flags[AXIS_IDX].parameter);
    psr::parseColor(fill_color, flags[COLOR_IDX].parameter);
    getFinFoutNames(input_file_name, output_file_name);
    getSecondFinName(second_file_name);

    ie::ImageBMP image_bmp;
    image_bmp.readImageFromFile(input_file_name.c_str());

    ie::ImageBMP second_image_bmp;
```

```
    second_image_bmp.readImageFromFile(second_file_name.c_str());

    image_bmp.concatBMP(second_image_bmp, axis_idx, fill_color);
    image_bmp.writeImageToFile(output_file_name.c_str());
}

void hdlr::Handler::handleFlags()
{
    if (isHelp()) {
        doHelp();
    } else if (isInfo()) {
        doInfo();
    } else if (isBorderRectangles()) {
        doBorderRectangles();
    } else if (isCircle()) {
        doCircle();
    } else if (isRGBFilter()) {
        doRGBFilter();
    } else if (isSplit()) {
        doSplit();
    } else if (isConcat()){
        doConcat();
    } else {
        throwError("Error: invalid set of flags.", HANDLER_ERROR);
    }
}
```

Название файла: parser.h

```
#ifndef PARSER
#define PARSER

#include "imageBMP.h"

#include <string>
#include <functional>

#define PARSER_ERROR 46


namespace psr
{
void throwError(const char *message, int exit_code);

void parseCoords(int& x, int& y, std::string& str);
void parseNumber(int& number, std::string& str);
void parseColor(ie::Color& color, std::string& str);
void parseComponentName(int& parameter, std::string& str);
void parseAxis(int& parameter, std::string& str);

void checkValueValidity(int value, std::function<bool(int)> check_func);
}

#endif
```

Название файла: parser.cpp

```
#include "imageBMP.h"
#include "parser.h"

#include <string>
#include <string.h>
#include <regex.h>
#include <functional>

void psr::throwError(const char *message, int exit_code)
{
    printf("%s\n", message);
    exit(exit_code);
}

void psr::parseCoords(int& x, int& y, std::string& str)
{
    regex_t rx;
    regcomp(&rx, "(-?[0-9]+)\\.(-?[0-9]+)", REG_EXTENDED);

    regmatch_t groups[3];

    if (regexec(&rx, str.c_str(), 3, groups, 0) != 0) {
        throwError("Error: coords parsing failed.", PARSER_ERROR);
    }

    std::string string;

    for (int i = groups[1].rm_so; i < groups[1].rm_eo; i++) {
          string += str[i];
     }
    x = atoi(string.c_str());

    string.clear();
    for (int i = groups[2].rm_so; i < groups[2].rm_eo; i++) {
          string += str[i];
     }
    y = atoi(string.c_str());
}

void psr::parseNumber(int& number, std::string& str)
{
    regex_t rx;
    regcomp(&rx, "-?[0-9]+", REG_EXTENDED);

    if (regexec(&rx, str.c_str(), 0, NULL, 0) != 0) {
        throwError("Error: number parsing failed.", PARSER_ERROR);
    }

    number = atoi(str.c_str());
}


void psr::parseColor(ie::Color& color, std::string& str)
{
    regex_t rx;
    regcomp(&rx, "([0-9]+)\\.([0-9]+)\\.([0-9]+)", REG_EXTENDED);
```

```
    regmatch_t groups[4];

    if (regexec(&rx, str.c_str(), 4, groups, 0) != 0) {
        throwError("Error: color parsing failed.", PARSER_ERROR);
    }

    std::string string;

    for (int i = groups[1].rm_so; i < groups[1].rm_eo; i++) {
          string += str[i];
     }
    color.r = atoi(string.c_str());

    string.clear();
    for (int i = groups[2].rm_so; i < groups[2].rm_eo; i++) {
          string += str[i];
     }
    color.g = atoi(string.c_str());

    string.clear();
    for (int i = groups[3].rm_so; i < groups[3].rm_eo; i++) {
          string += str[i];
     }
    color.b = atoi(string.c_str());
}

void psr::parseComponentName(int& parameter, std::string& str)
{
    if (str == "red") {
        parameter = R_IDX;
        return;
    }
    if (str == "green") {
        parameter = G_IDX;
        return;
    }
    if (str == "blue") {
        parameter = B_IDX;
        return;
    }
    throwError("Error: component parsing failed.", PARSER_ERROR);
}

void    psr::checkValueValidity(int    value,    std::function<bool(int)>
check_func)
{
    if (!check_func(value)) {
        throwError("Error: value parsing failed.", PARSER_ERROR);
    }
}

void psr::parseAxis(int& parameter, std::string& str)
{
    if (str == "x") {
        parameter = X_IDX;
        return;
    }
```

```
    if (str == "y") {
        parameter = Y_IDX;
        return;
    }
    throwError("Error: axis parsing failed.", PARSER_ERROR);
}
```

Название файла: imageBMP.h

```
#ifndef IMAGE_BMP
#define IMAGE_BMP

#include "structures.h"

#include <stdio.h>
#include <vector>
#include <string.h>
#include <algorithm>
#include <stdlib.h>
#include <queue>
#include <limits.h>

#define VERTICAL                    0
#define HORIZONTAL                  1
#define FILE_ERROR                  40
#define BMP_PROCESSING_ERROR        41

namespace ie
{

void throwError(const char *message, int exit_code);

bool checkFileFormat(unsigned char *signature, unsigned int compression,
    unsigned short bits_per_pixel);

class ImageBMP
{
public:
    ImageBMP();

    ~ImageBMP();

    int getWidth();

    int getHeight();

    void readImageFromFile(const char *input_file_name);

    void showImageInfo();

    void writeImageToFile(const char *output_file_name);

    void clear();

    void setSize(int width, int height);

    ImageBMP copy(int x0, int y0, int x1, int y1);
```

```
    Color getColor(int x, int y);

    void setColor(int x, int y, Color color);

    void drawLine(int x0, int y0, int x1, int y1,
        int thickness, Color color);

    void drawBresenhamCircle(int x0, int y0, int radius, Color color);

    void drawCircle(int x0, int y0, int radius, int thickness,
        Color color, bool fill, Color fill_color);

    void rgbFilter(int component_idx, int component_value);

    void concatBMP(ImageBMP& second_image_bmp, int axis, Color
fill_color);

    void borderRectangles(Color rectangles_color, Color border_color, int
thickness);

    bool checkCoordsValidity(int x, int y);

private:
    BMPHeader           bmph;
    DIBHeader           dibh;
    int                 Width;
    int                 Height;
    RGB                 **bitmap;

    void allocateMemmoryForPixels();

    void freeMemmoryForPixels();

    void clearPixels();

    void drawLineHigh(int x0, int y0, int x1, int y1,
        int thickness, Color color);

    void drawLineLow(int x0, int y0, int x1, int y1,
        int thickness, Color color);
};

}
#endif
```

Название файла: imageBMP.cpp

```
#include "imageBMP.h"

void ie::throwError(const char *message, int exit_code)
{
    printf("%s\n", message);
    exit(exit_code);
}

ie::ImageBMP::ImageBMP() = default;
```

```
ie::ImageBMP::~ImageBMP() = default;

int ie::ImageBMP::getWidth()
{
    return Width;
}

int ie::ImageBMP::getHeight()
{
    return Height;
}

void ie::ImageBMP::allocateMemmoryForPixels()
{
    bitmap = (RGB**)malloc(Height*sizeof(RGB*));

    for (int i = 0; i < Height; i++) {
        bitmap[i] = (RGB*)malloc((Width*sizeof(RGB)+3)&(-4));
    }
}

void ie::ImageBMP::freeMemmoryForPixels()
{
    for (int i = 0; i < Height; i++) {
        free(bitmap[i]);
    }
    free(bitmap);
}

void ie::ImageBMP::clearPixels()
{
   for (int i = 0; i < Height; i++) {
        for(int j = 0; j < Width; j++){
            bitmap[i][j].r = 0;
            bitmap[i][j].g = 0;
            bitmap[i][j].b = 0;
        }
    }
}

bool   ie::checkFileFormat(unsigned   char   *signature,   unsigned   int
compression,
    unsigned short bits_per_pixel)
{
    return ((signature[0] == 0x4d && signature[1] == 0x42) || (signature[0]
== 0x42 && signature[1] == 0x4d))
        && compression == 0 && bits_per_pixel == 24;
}

void ie::ImageBMP::showImageInfo()
{
    printf("BMPHeader\n");
    printf("Signature:\t%x\n",*((short int*)bmph.signature));
    printf("File size:\t%u\n",bmph.file_size);
    printf("Reserved 1:\t%x\n",*((short int*)bmph.reserved1));
    printf("Reserved 2:\t%x\n",*((short int*)bmph.reserved2));
```

```
    printf("Pixel offset:\t%u\n", bmph.pixel_offset);

    printf("DIBHeader\n");
    printf("Header size:\t%u\n",dibh.byte_count);
    printf("Width:\t%u\n",dibh.width);
    printf("Height:\t%u\n",dibh.height);
    printf("Color planes:\t%hu\n",dibh.color_planes);
    printf("Bits per pixel:\t%hu\n",dibh.bits_per_pixel);
    printf("Compression:\t%u\n",dibh.compression);
    printf("Image size:\t%u\n",dibh.image_size);
    printf("yPixels per meter:\t%u\n",dibh.pwidth);
    printf("xPixels per meter:\t%u\n",dibh.pheight);
    printf("Colors in color table:\t%u\n",dibh.color_count);
    printf("Important color count:\t%u\n",dibh.important_color_count);
}

void ie::ImageBMP::readImageFromFile(const char *input_file_name)
{
    FILE* fin = fopen(input_file_name, "rb");

    if (!fin) {
        throwError("Error: file could not be opened.", FILE_ERROR);
    }

    fread(&bmph, sizeof(BMPHeader), 1, fin);
    fread(&dibh, sizeof(DIBHeader), 1, fin);

    if (!checkFileFormat(bmph.signature, dibh.compression,
        dibh.bits_per_pixel)) {
        fclose(fin);
        throwError("Error: wrong file format.", FILE_ERROR);
    }

    fseek(fin, bmph.pixel_offset, SEEK_SET);

    Width = (int)(dibh.width);
    Height = (int)(dibh.height);

    allocateMemmoryForPixels();

    for (int i = 0; i < Height; i++) {
        fread(bitmap[Height -i -1], 1, (Width*sizeof(RGB)+3)&(-4), fin);
    }

    fclose(fin);
}

void ie::ImageBMP::writeImageToFile(const char *output_file_name)
{
    FILE* fout = fopen(output_file_name, "wb");

    dibh.width = (unsigned int)Width;
    dibh.height = (unsigned int)Height;

    fwrite(&bmph, 1, sizeof(BMPHeader), fout);
    fwrite(&dibh, 1, sizeof(DIBHeader), fout);
```

```
    fseek(fout, bmph.pixel_offset, SEEK_SET);

    for (int i = 0; i < Height; i++) {
        fwrite(bitmap[Height -i -1], 1, (Width*sizeof(RGB)+3)&(-4), fout);
    }

    freeMemmoryForPixels();

    fclose(fout);
}

bool ie::ImageBMP::checkCoordsValidity(int x, int y)
{
    return (x >= 0 && x < Width && y >= 0 && y < Height);
}

void ie::ImageBMP::clear()
{
    clearPixels();
}

ie::Color ie::ImageBMP::getColor(int x, int y)
{
    if (!checkCoordsValidity(x, y)) {
        return {0, 0, 0};
    }
    ie:Color color;
    color.r = bitmap[y][x].r;
    color.g = bitmap[y][x].g;
    color.b = bitmap[y][x].b;

    return color;
}

void ie::ImageBMP::setColor(int x, int y, Color color)
{
    if (!checkCoordsValidity(x, y)) {
        return;
    }
    bitmap[y][x].r = color.r;
    bitmap[y][x].g = color.g;
    bitmap[y][x].b = color.b;
}

bool checkOnCircleLine(int x, int y, int x0, int y0, int radius, int
thickness)
{
    bool     flag1     =     (x-x0)*(x-x0)     +     (y-y0)*(y-y0)     <=
(radius+thickness/2)*(radius+thickness/2);
    bool flag2 = (x-x0)*(x-x0) + (y-y0)*(y-y0) >= (std::max(0, radius-
thickness/2))*(std::max(0, radius-thickness/2));
    return flag1 && flag2;
}

bool checkInCircle(int x, int y, int x0, int y0, int radius, int thickness)
{
```

```
    bool flag = (x-x0)*(x-x0) + (y-y0)*(y-y0) <= (radius-
thickness/2)*(radius-thickness/2);
    return flag;
}

void ie::ImageBMP::drawBresenhamCircle(int x0, int y0, int radius, Color
color)
{
    int D = 3 - 2 * radius;
    int x = 0;
    int y = radius;
    while (x <= y) {
        setColor(x+x0, y+y0, color);
        setColor(y+x0, x+y0, color);
        setColor(-y+x0, x+y0, color);
        setColor(-x+x0, y+y0, color);
        setColor(-x+x0, -y+y0, color);
        setColor(-y+x0, -x+y0, color);
        setColor(y+x0, -x+y0, color);
        setColor(x+x0, -y+y0, color);

        if (D < 0) {
            D += 4 * x + 6;
            x++;
        } else {
            D += 4 * (x - y) + 10;
            x++;
            y--;
        }
    }
}

void ie::ImageBMP::drawCircle(int x0, int y0, int radius, int thickness,
    Color color, bool fill, Color fill_color)
{
    for (int y = std::max(0, y0-radius-thickness/2); y <= std::min(Height-
1, y0+radius+thickness/2); y++) {
        for (int x = std::max(0, x0-radius-thickness/2); x <=
std::min(Width-1, x0+radius+thickness/2); x++) {
            if (fill && checkInCircle(x, y, x0, y0, radius, thickness)) {
                setColor(x, y, fill_color);
            }
            if (checkOnCircleLine(x, y, x0, y0, radius, thickness)) {
                setColor(x, y, color);
            }
        }
    }

    drawBresenhamCircle(x0, y0, radius-thickness/2, color);
    drawBresenhamCircle(x0, y0, radius+thickness/2, color);
}

void ie::ImageBMP::drawLineLow(int x0, int y0, int x1, int y1,
    int thickness, Color color)
{
    int dx = x1 - x0;
    int dy = y1 - y0;
```

```
    int yi = 1;
    if (dy < 0) {
        yi = -1;
        dy = -dy;
    }
    int D = (2 * dy) - dx;
    int y = y0;
    for (int x = x0; x <= x1; x++) {
        if (thickness == 1) {
            setColor(x, y, color);
        } else {
            drawCircle(x, y, thickness/2, 1, color, true, color);
        }

        if (D > 0) {
            y += yi;
            D += 2 * (dy - dx);
        } else {
            D += 2 * dy;
        }
    }
}

void ie::ImageBMP::drawLineHigh(int x0, int y0, int x1, int y1,
    int thickness, Color color)
{

    int dx = x1 - x0;
    int dy = y1 - y0;
    int xi = 1;
    if (dx < 0) {
        xi = -1;
        dx = -dx;
    }
    int D = (2 * dx) - dy;
    int x = x0;
    for (int y = y0; y <= y1; y++) {
        if (thickness == 1) {
            setColor(x, y, color);
        } else {
            drawCircle(x, y, thickness/2, 1, color, true, color);
        }

        if (D > 0) {
            x += xi;
            D += 2 * (dx - dy);
        } else {
            D += 2 * dx;
        }
    }
}

void ie::ImageBMP::drawLine(int x0, int y0, int x1, int y1,
    int thickness, Color color)
{
    if (abs(y1 - y0) < abs(x1 - x0)) {
        if (x0 > x1) {
```

```
            std::swap(x0, x1);
            std::swap(y0, y1);
        }
        drawLineLow(x0, y0, x1, y1, thickness, color);
    } else {
        if (y0 > y1) {
            std::swap(x0, x1);
            std::swap(y0, y1);
        }
        drawLineHigh(x0, y0, x1, y1, thickness, color);
    }
}

void ie::ImageBMP::setSize(int width, int height)
{
    Width = width;
    Height = height;

    allocateMemmoryForPixels();

    clear();
}

ie::ImageBMP ie::ImageBMP::copy(int x0, int y0, int x1, int y1)
{
    if (x0 > x1) {
        std::swap(x0, x1);
    }
    if (y0 > y1) {
        std::swap(y0, y1);
    }

    ImageBMP copy_image;
    copy_image.setSize(x1-x0+1, y1-y0+1);
    for (int y = 0; y < copy_image.getHeight(); y++) {
        for (int x = 0; x < copy_image.getWidth(); x++) {
            copy_image.setColor(x, y, getColor(x + x0, y + y0));
        }
    }
    return copy_image;
}

void    ie::ImageBMP::borderRectangles(Color    rectangles_color,    Color
border_color, int thickness)
{
    int current_area, max_current_area;
    int a, b;
    int x0, y0, x1, y1;
    int a_max, b_max;
    int already_found;
    std::vector<Coord> found_rectangles;

    for(int x = 0; x < Width; x++) {
        for(int y = 0; y < Height; y++) {
            already_found = 0;
            for (int i = 0; i < found_rectangles.size(); i+=2) {
```

```
                if    (found_rectangles[i].x    <=    x    &&    x    <=
found_rectangles[i+1].x
                    &&    found_rectangles[i].y    <=    y    &&    y    <=
found_rectangles[i+1].y) {
                        already_found = 1;
                        break;
                    }
            }

            if (already_found) {
                continue;
            }

            if(getColor(x, y) == rectangles_color && getColor(x-1, y) !=
rectangles_color && getColor(x, y-1) != rectangles_color){
                b_max = Height;
                max_current_area = 0;
                a = 0, b = 0;
                while((x+a    <    Width)    &&    getColor(x+a,    y)    ==
rectangles_color){
                    while((y+b   <   Height)   &&   getColor(x+a,   y+b)   ==
rectangles_color){
                        current_area += a+1;
                        b++;
                        if(b > b_max)
                            break;
                    }
                    if(current_area > max_current_area){
                        max_current_area = current_area;
                        a_max = a;
                        b_max = b - 1;
                    }
                    current_area = 0;
                    b = 0;
                    a++;
                }

                x0 = x, y0 = y;
                x1 = x+a_max, y1 = y+b_max;

                found_rectangles.push_back({x0, y0});
                found_rectangles.push_back({x1, y1});

                drawLine(x0, y0, x1, y0, thickness, border_color);
                drawLine(x0, y0, x0, y1, thickness, border_color);
                drawLine(x0, y1, x1, y1, thickness, border_color);
                drawLine(x1, y0, x1, y1, thickness, border_color);
            }
        }
    }
}

void ie::ImageBMP::concatBMP(ImageBMP& second_image_bmp, int axis, Color
fill_color)
{
    int   new_width,   new_height,   max_width,   max_height,   min_width,
min_height;
```

```
    max_width = std::max(Width, second_image_bmp.getWidth());
    max_height = std::max(Height, second_image_bmp.getHeight());

    min_width = std::min(Width, second_image_bmp.getWidth());
    min_height = std::min(Height, second_image_bmp.getHeight());

    if (axis == 0) {
        new_width = Width + second_image_bmp.getWidth();
        new_height = max_height;
    } else {
        new_width = max_width;
        new_height = Height + second_image_bmp.getHeight();
    }

    ImageBMP copy_image = copy(0, 0, Width-1, Height-1);

    setSize(new_width, new_height);

    if (axis == 0) {
        int new_x;
        for (int y = 0; y < max_height; y++, new_x = 0) {
            for (int x = 0; x < min_width; x++, new_x += 2) {
                if (copy_image.checkCoordsValidity(x, y)) {
                    setColor(new_x, y, copy_image.getColor(x, y));
                } else {
                    setColor(new_x, y, fill_color);
                }

                if (second_image_bmp.checkCoordsValidity(x, y)) {
                    setColor(new_x + 1, y, second_image_bmp.getColor(x,
y));
                } else {
                    setColor(new_x + 1, y, fill_color);
                }
            }
            for (int x = min_width; x < max_width; x++, new_x++){
                if (copy_image.checkCoordsValidity(x, y)) {
                    setColor(new_x, y, copy_image.getColor(x, y));
                } else if (second_image_bmp.checkCoordsValidity(x, y)) {
                    setColor(new_x, y, second_image_bmp.getColor(x, y));
                } else {
                    setColor(new_x, y, fill_color);
                }
            }
        }
    } else {
        int new_y;
        for (int x = 0; x < max_width; x++, new_y = 0) {
            for (int y = 0; y < min_height; y++, new_y += 2) {
                if (copy_image.checkCoordsValidity(x, y)) {
                    setColor(x, new_y, copy_image.getColor(x, y));
                } else {
                    setColor(x, new_y, fill_color);
                }

                if (second_image_bmp.checkCoordsValidity(x, y)) {
```

```
                    setColor(x, new_y + 1, second_image_bmp.getColor(x,
y));
                } else {
                    setColor(x, new_y + 1, fill_color);
                }
            }
            for (int y = min_height; y < max_height; y++, new_y++){
                if (copy_image.checkCoordsValidity(x, y)) {
                    setColor(x, new_y, copy_image.getColor(x, y));
                } else if (second_image_bmp.checkCoordsValidity(x, y)) {
                    setColor(x, new_y, second_image_bmp.getColor(x, y));
                } else {
                    setColor(x, new_y, fill_color);
                }
            }
        }
    }
}

void ie::ImageBMP::rgbFilter(int component_idx, int component_value)
{
    if (component_idx == R_IDX) {
        for (int y = 0; y < Height; y++) {
            for (int x = 0; x < Width; x++) {
                bitmap[y][x].r = component_value;
            }
        }
    } else if (component_idx == G_IDX) {
        for (int y = 0; y < Height; y++) {
            for (int x = 0; x < Width; x++) {
                bitmap[y][x].g = component_value;
            }
        }
    } else if (component_idx == B_IDX) {
        for (int y = 0; y < Height; y++) {
            for (int x = 0; x < Width; x++) {
                bitmap[y][x].b = component_value;
            }
        }
    }
}
```

Название файла: structures.h

```
#ifndef STRUCTURES
#define STRUCTURES

#define R_IDX        0
#define G_IDX        1
#define B_IDX        2

#define X_IDX        0
#define Y_IDX        1

#define SIG_BYTES          2
#define RESERVED_1_BYTES   2
#define RESERVED_2_BYTES   2
```

```
namespace ie
{
struct Coord
{
    int x;
    int y;
};

struct Color
{
    int r;
    int g;
    int b;

    bool operator==(Color other)
    {
        return (r == other.r) &&
               (g == other.g) &&
               (b == other.b);
    }

    bool operator!=(Color other)
    {
        return !(*this == other);
    }

    void inverse()
    {
        r = 255 - r;
        g = 255 - g;
        b = 255 - b;
    }

    void gray()
    {
        r = (0.299 * r) + (0.587 * g) + (0.114 * b);
        g = (0.299 * r) + (0.587 * g) + (0.114 * b);
        b = (0.299 * r) + (0.587 * g) + (0.114 * b);
    }
};

#pragma pack(push, 1)
struct BMPHeader
{
    unsigned char       signature[SIG_BYTES];
    unsigned int        file_size;
    unsigned char       reserved1[RESERVED_1_BYTES];
    unsigned char       reserved2[RESERVED_2_BYTES];
    unsigned int        pixel_offset;
};
struct DIBHeader
{
    unsigned int       byte_count;
    unsigned int       width;
    unsigned int       height;
    unsigned short     color_planes;
```

```
    unsigned short     bits_per_pixel;
    unsigned int       compression;
    unsigned int       image_size;
    unsigned int       pwidth;
    unsigned int       pheight;
    unsigned int       color_count;
    unsigned int       important_color_count;
};
struct RGB
{
    unsigned char b;
    unsigned char g;
    unsigned char r;
};
#pragma pack(pop)

}
#endif
```

Название файла: Makefile

```
all : main.o parser.o handler.o libimageBMP.so
      g++ main.o parser.o handler.o -fPIC -Wl,-rpath=. -L. -limageBMP -o
cw

main.o : main.cpp handler.h
      g++ main.cpp -c -Wall

parser.o : parser.cpp parser.h imageBMP.h
      g++ parser.cpp -c -Wall

handler.o : handler.cpp handler.h parser.h
      g++ handler.cpp -c -Wall

libimageBMP.so : imageBMP.cpp imageBMP.h
      g++ imageBMP.cpp -fPIC -shared -o libimageBMP.so

clean :
      -rm cw libimageBMP.so *.o
```

ПРИЛОЖЕНИЕ В

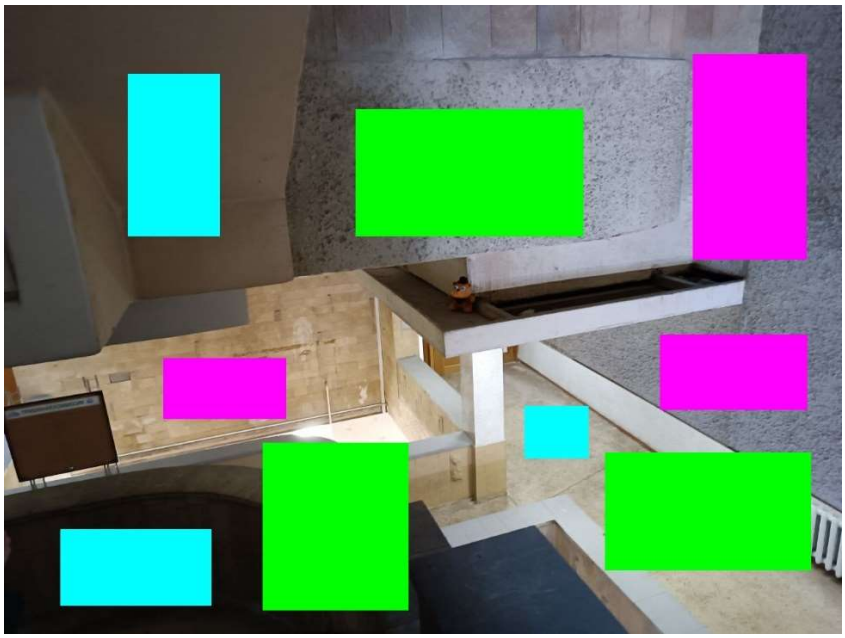
ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ

Команда help:

```
lastikp0@lastikp0-PC:~/Shapovalenko_Egor_cw/src$ ./cw --help
Course work for option 5.8, created by Egor Shapovalenko
--help (-h)             Prints the help
--input (-i)            Sets the name of the input image. If the flag is omitted, it is assumed that the name of the input image is passed as the last argument
--output (-o)           Sets the name of the output image. If the flag is omitted, it is assumed that the name of the input image is out.bmp
--info                  Prints information about the image

Basic flags:
--thickness             Line thickness. Format: NUMBER
--color                 Line color. Format: RED.GREEN.BLUE
--fill                  Works as a binary value: there is a flag - true, there is no flag - false
--fill color            Fill color. Can be used without --fill flag. Format: RED.GREEN.BLUE

--filled rects          Drawing a border for all rectangles filled with specified color. Required: --color, --border color, --thickness
--color                 Color that rectangles are filled with. Format: RED.GREEN.BLUE
--border color          Color of the border. Format: RED.GREEN.BLUE

--circle                Drawing a circle. Required: --center, --radius, --thickness, --color. Optional: --fill, --fill color
--center                Coordinates of the center. Format: x.y
--radius                Radius. Format: NUMBER

--rgbfilter             RGB filter component
--component name        Which component needs to be changed. Format: {red || green || blue}
--component value       To which value it needs to be changed. Format: NUMBER

--split                 Divides the image into number y*number x parts. Required: --thickness, --color
--number x              The number of parts along the Y-axis. Format: NUMBER
--number y              The number of parts along the X-axis. Format: NUMBER
```

Команда info:

Исходный файл:



Результат:

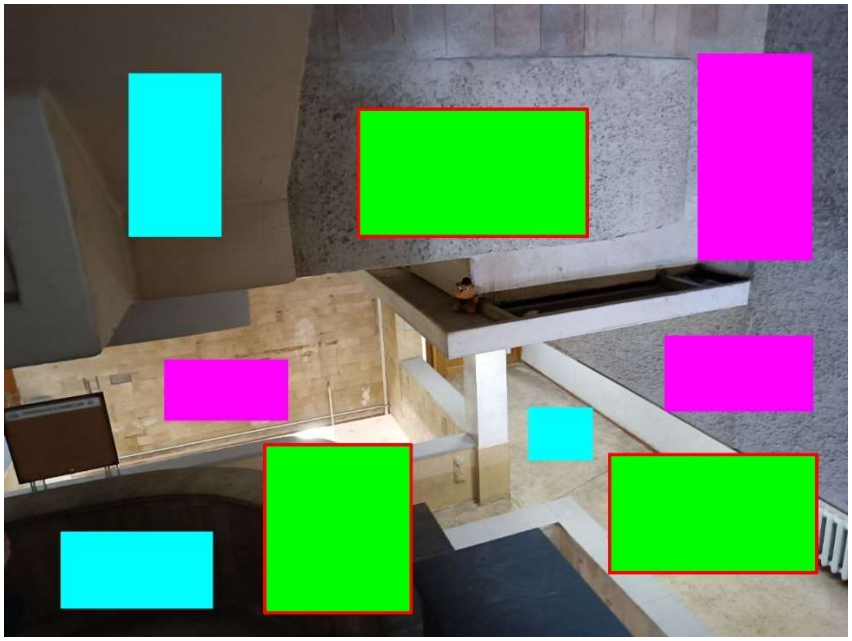
```
lastikp0@lastikp0-PC:~/Shapovalenko_Egor_cw/src$ ./cw --info ./1.bmp
Course work for option 5.8, created by Egor Shapovalenko
BMPHeader
Signature:      4d42
File size:      3686538
Reserved 1:     0
Reserved 2:     0
Pixel offset:   138

DIBHeader
Header size:    124
Width:  1280
Height: 960
Color planes:   1
Bits per pixel: 24
Compression:    0
Image size:     3686400
yPixels per meter:      2835
xPixels per meter:      2835
Colors in color table:  0
Important color count:  0
```

Команда filled_rects:

Исходный файл:

Результат:

```
lastikp0@lastikp0-PC:~/Shapovalenko_Egor_cw/src$ ./cw --filled rects --color 0.255.0 --border color 255.0.0 --thickness 5 ./1.bmp
Course work for option 5.8, created by Egor Shapovalenko
```

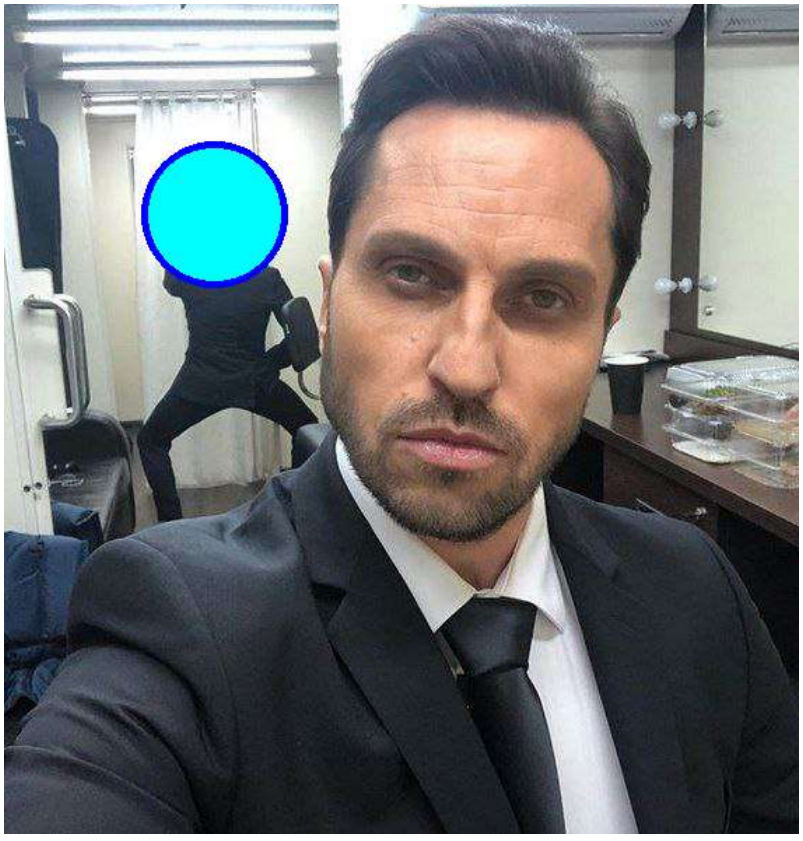


Команда circle:

Исходный файл:

Результат:

```
lastikp0@lastikp0-PC:~/Shapovalenko_Egor_cw/src$ ./cw --circle --center 150.150 --radius 50 --thickness 5 --color 0.0.255 --fill --fill color 0.255.255 ./2.bmp
Course work for option 5.8, created by Egor Shapovalenko
```

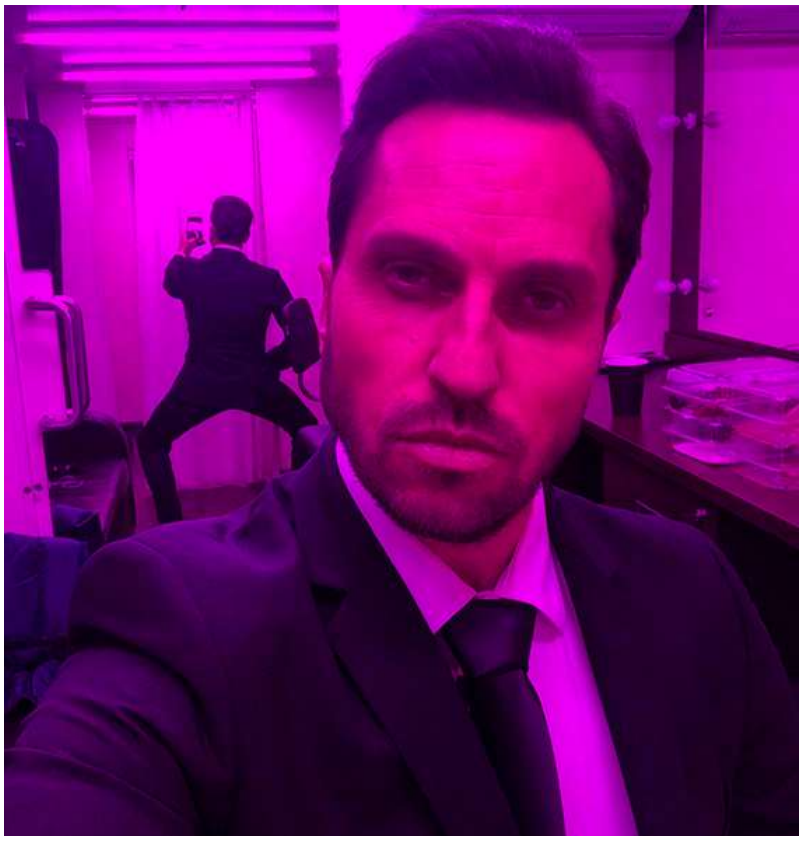


Команда rgbfilter:

Исходный файл:

Результат:

```
lastikp0@lastikp0-PC:~/Shapovalenko_Egor_cw/src$ ./cw --rgbfilter --component name green --component value 0 ./2.bmp
Course work for option 5.8, created by Egor Shapovalenko
```

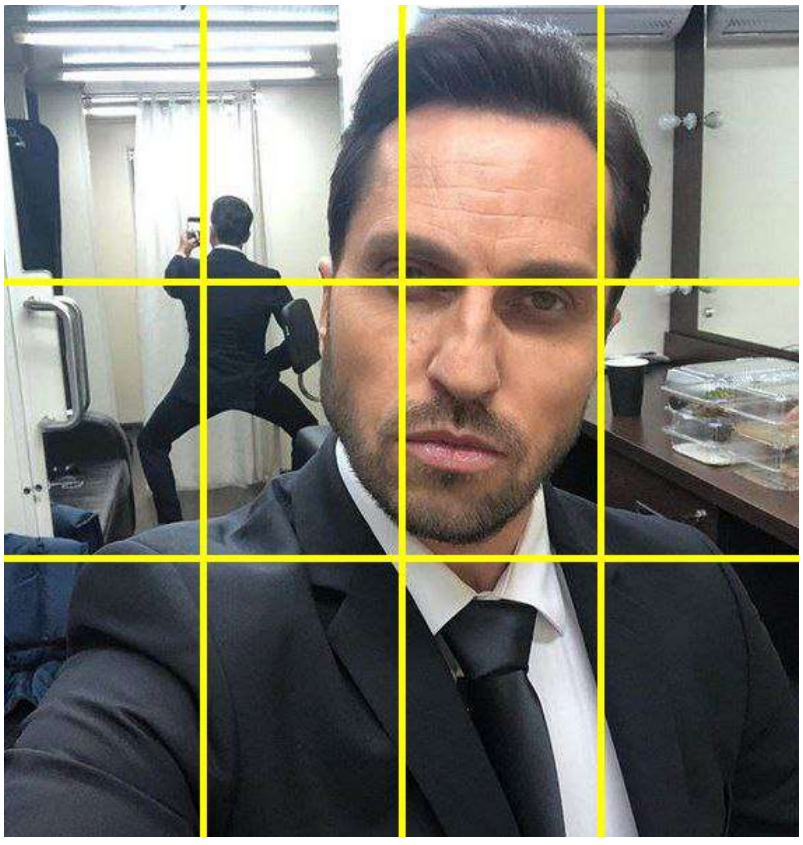


Команда split:

Исходный файл:

Результат:

```
lastikp0@lastikp0-PC:~/Shapovalenko_Egor_cw/src$ ./cw --split --number x 4 --number y 3 --thickness 5 --color 255.255.0 ./2.bmp
Course work for option 5.8, created by Egor Shapovalenko
```



Неправильный флаг:

```
lastikp0@lastikp0-PC:~/Shapovalenko_Egor_cw/src$ ./cw --test falg ./1.bmp
Course work for option 5.8, created by Egor Shapovalenko
Error: wrong flag.
```

Неправильный набор флагов:

```
lastikp0@lastikp0-PC:~/Shapovalenko_Egor_cw/src$ ./cw --info --thickness 5 --color 0.0.0 ./2.bmp
Course work for option 5.8, created by Egor Shapovalenko
Error: invalid set of flags.
```

Неправильный формат координат:

```
lastikp0@lastikp0-PC:~/Shapovalenko_Egor_cw/src$ ./cw --circle --center .1 --radius 5 --thickness 1 --color 0.0.0 ./2.bmp
Course work for option 5.8, created by Egor Shapovalenko
Error: coords parsing failed.
```

Неправильный формат числа:

```
lastikp0@lastikp0-PC:~/Shapovalenko_Egor_cw/src$ ./cw --circle --center 10.10 --radius rad --thickness 1 --color 0.0.0 ./2.bmp
Course work for option 5.8, created by Egor Shapovalenko
Error: number parsing failed.
```

Неправильный формат цвета:

```
lastikp0@lastikp0-PC:~/Shapovalenko_Egor_cw/src$ ./cw --circle --center 10.10 --radius 5 --thickness 1 --color cyan ./2.bmp
Course work for option 5.8, created by Egor Shapovalenko
Error: color parsing failed.
```

Неправильное значение:

```
lastikp0@lastikp0-PC:~/Shapovalenko_Egor_cw/src$ ./cw --circle --center 10.10 --radius 0 --thickness 1 --color 0.0.0 ./2.bmp
Course work for option 5.8, created by Egor Shapovalenko
Error: value parsing failed.
```

Неправильный формат компоненты:

```
lastikp0@lastikp0-PC:~/Shapovalenko_Egor_cw/src$ ./cw --rgbfilter --component name r --component value 0 ./2.bmp
Course work for option 5.8, created by Egor Shapovalenko
Error: component parsing failed.
```

Невозможно открыть файл:

```
lastikp0@lastikp0-PC:~/Shapovalenko_Egor_cw/src$ ./cw --rgbfilter --component name red --component value 0 ./2.bmp
Course work for option 5.8, created by Egor Shapovalenko
Error: file could not be opened.
```

Неправильный формат файла:

```
lastikp0@lastikp0-PC:~/Shapovalenko_Egor_cw/src$ ./cw --rgbfilter --component name red --component value 0 ./1.jpeg
Course work for option 5.8, created by Egor Shapovalenko
Error: wrong file format.
```

```
lastikp0@lastikp0-PC:~/Shapovalenko_Egor_cw/src$ ./cw --rgbfilter --component name red --component value 0 ./1.bmp
Course work for option 5.8, created by Egor Shapovalenko
Error: wrong file format.
```