**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

Курсовая РАБОТА

**по дисциплине «Программирование»**

Тема: Обработка изображений

|  |  |  |
| --- | --- | --- |
| Студент гр. 3341 |  | Шаповаленко Е.В. |
| 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 | 7 |
| 1. | | Ход выполнения работы | 8 |
| 1.1. | | Структуры данных | 8 |
| 1.2. | | Ввод/вывод | 8 |
| 1.3 | | Проверка соблюдения формата ввода | 10 |
| 1.3. | | Обработка изображения | 10 |
| 1.4. | | Makefile | 12 |
|  | Заключение | 13 |
|  | Приложение А. Исходный код программы | 14 |
|  | Приложение B. Демонстрация работы программы | 40 |

**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 B.

**1. Ход выполнения работы**

* 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. **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. **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. **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. **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 Сoncatenates 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

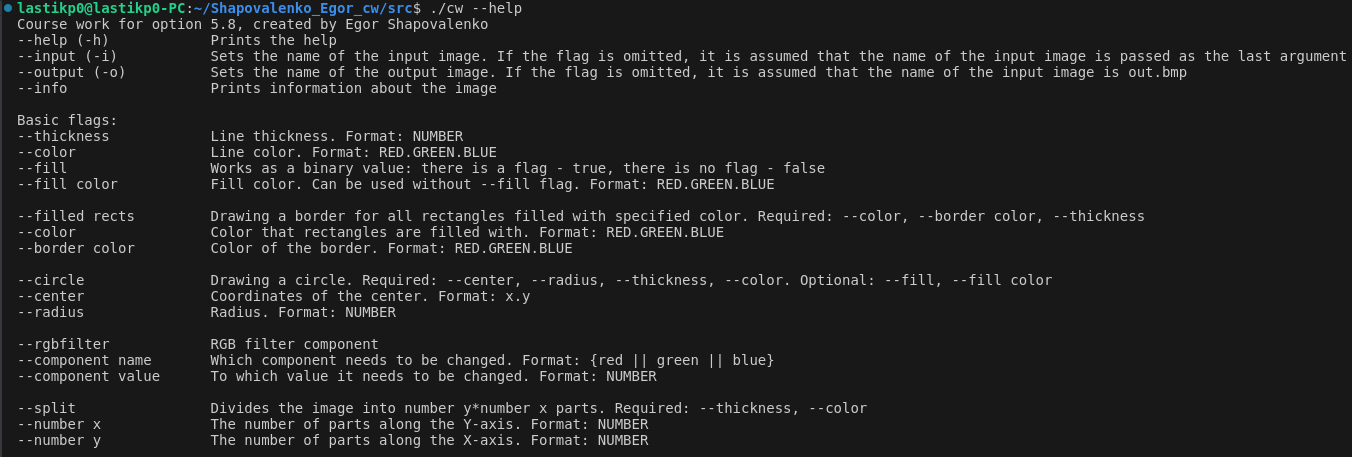
clean :

-rm cw libimageBMP.so \*.o

**приложение B**

**Демонстрация работы программы**

Команда help:

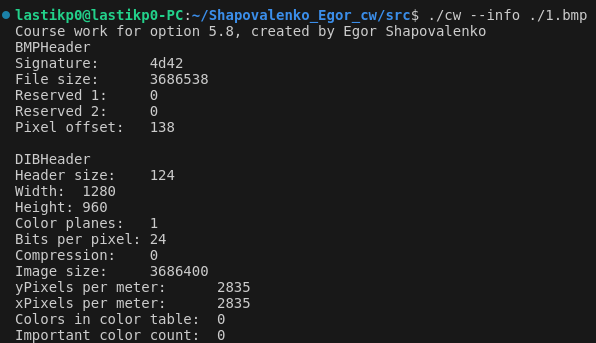


Команда info:

Исходный файл:



Результат:



Команда filled\_rects:

Исходный файл:



Результат:





Команда circle:

Исходный файл:



Результат:





Команда rgbfilter:

Исходный файл:



Результат:





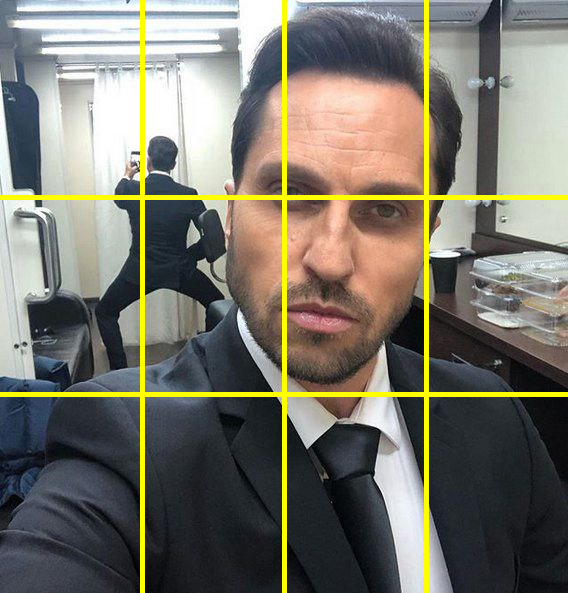
Команда split:

Исходный файл:



Результат:





Неправильный флаг:



Неправильный набор флагов:



Неправильный формат координат:



Неправильный формат числа:



Неправильный формат цвета:



Неправильное значение:



Неправильный формат компоненты:



Невозможно открыть файл:



Неправильный формат файла:



