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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Работа с bmp файлами

Студент гр. 7303 _____ Юсковец А.В.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2018

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ (РАБОТА С BMP ФАЙЛАМИ)

Студент Юсковец А.В.

Группа 7303

Тема работы: Работа с bmp файлами

Исходные данные:

- 24 бита на цвет, без сжатия
- файл всегда соответствует формату BMP (но стоит помнить, что версий у формата несколько)
- обратите внимание на выравнивание; мусорные данные, если их необходимо дописать в файл для выравнивания, должны быть нулями.
- обратите внимание на порядок записи пикселей
- все поля стандартных BMP заголовков в выходном файле должны иметь те же значения что и во входном (разумеется кроме тех, которые должны быть изменены).

Содержание пояснительной записки:

1. Аннотация; 2. Содержание; 3. Введение; 4. Bmp_image; 5. Bmp_image24;
7. BMP; 8. Qt классы; 9. Main Window; 10. Диалоговые окна; 11. Заключение;
12. Список использованных источников 13. Приложение А. Примеры работы

Дата выдачи задания: 19.02.2018

Дата сдачи курсовой работы:

Дата защиты курсовой работы:

Студент _____ Юсковец А.В.

Преподаватель _____ Берленко Т.А.

АННОТАЦИЯ

В работе описаны все классы, используемые для реализации требуемого функционала. Также были описаны классы, реализующие пользовательский интерфейс. К каждому описываемому классу приложен код и, если этот класс реализует интерфейс, рисунок соответствующего окна.

Описаны связи между классами, условия испуская сигналов Qt-объектами и места их обработки. В конце данной работы представлена вся структура проекта в виде приложения с исходным кодом каждого из файлов.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Программа должна реализовывать следующий функционал по обработке bmp-файла:

- (1) Инверсия цвета в заданной области. Функционал определяется:
    - Координатами левого верхнего угла области
    - Координатами правого нижнего угла области
- (2) Преобразовать в Ч/Б изображение (любым простым способом). Функционал определяется:
    - Координатами левого верхнего угла области
    - Координатами правого нижнего угла области
    - Алгоритмом, если реализовано несколько алгоритмов преобразования изображения (по желанию студента)
- (3) Изменение размера изображения с его обрезкой или расширением фона. Функционал определяется:
    - Действием: увеличение или уменьшение изображения
    - Цветом фона при увеличении изображения
    - Точкой относительно которой производится действие: центр, левый верхний, правый верхний, левый нижний, правый нижний угол.

Все функции, перечисленные раннее были реализованы без использования предоставляемого Qt функционала.

Для реализации был написан класс, хранящий в себе представление bmp файла. Каждая функция вынесена в отдельный public метод. Все `выравнивающие байты` заполняются нулями.

1. BMP_IMAGE

Данный класс является абстрактным. Он послужит родителем для класс Bmp_image24, в котором будет реализован весь требуемый функционал.

1.1. Структуры bmp-файла.

В объявлении структур была использована директива препроцессора #pragma pack(push, 1) чтобы выровнять поля структуры по одному байту. Это понадобится при считывании данных структур из файла.

1.1.1. Структура для заголовока файла.

```
#pragma pack(push, 1)
struct BitMapFileHeader {
    char           bfType1;
    char           bfType2;
    unsigned       bfSize;
    unsigned short bfReserved1;
    unsigned short bfReserved2;
    unsigned       bfOffBits;
};
```

bfType1 — поле для буквы B

bfType2 — поле для буквы M

bfSize — размер всего файла

bfReserved1, bfReserved2 — зарезервированные байты

bfOffBits — порядковый номер байта, с которого начинается непосредственно изображение.

1.1.2. Структура для информационных полей файла.

```
struct BitMapInfo {
    unsigned       biSize;
    int            biWidth;
    int            biHeight;
    unsigned short biPlanes;
    unsigned short biBitCount;
    unsigned       biCompression;
    unsigned       biSizeImage;
    int            biXPelsPerMeter;
    int            biYPelsPerMeter;
    unsigned       biClrUsed;
    unsigned       biClrImportant;
};
```

```
#pragma pack(pop)
```

biSize — размер данной структуры в байтах

biWidth — ширина изображения в пикселях

biHeight — высота изображения в пискелях

biPlanes — всегда заполнено 1.

biBitCount — количество битов на цвет

biCompression — байт, отвечающий за сжатие изображения

biSizeImage — непосредственно размер изображения в байтах

biXPelsPerMeter — плотность пикселей по оси абсцисс

biYPelsPerMeter — плотность пикселей по оси ординат

biCrlUsed — размер палитры в ячейках (в данном задании не предусматривается наличие палитры у изображения)

biCrlImportant — количество ячеек от начала палитры до конца, включая последнюю ячейку.

1.2. Перечисление Resize_direction

Данное пространство имен будет использовано далее для базовых функций работы с bmp-файлами (создание, сохранение, открытие).

```
namespace Bmp {
    enum class Resize_direction {center, upper_left, upper_right, lower_right, lower_left};
}
```

center — отвечает за изменение изображения относительно центра

upper_left — отвечает за изменение изображения относительно верхнего левого угла

upper_right — отвечает за изменение изображения относительно верхнего правого угла

lower_right — отвечает за изменение изображения относительно нижнего правого угла

lower_left — отвечает за изменение изображения относительно нижнего левого угла.

1.3. Виртуальные методы

Каждый объявленный здесь метод будет описан далее в описание реализации оных для класса Bmp_image24.

```
class Bmp_image {
public:
    virtual ~Bmp_image() {}

    virtual void set_color(int x, int y, QColor const& color) = 0;

    virtual int get_size() const        = 0;

    virtual int get_height() const      = 0;

    virtual int get_width() const       = 0;

    virtual short get_bitcount() const  = 0;

    virtual uint8_t* get_raster() const = 0;

    virtual uint8_t get_raster(int x, int y) const = 0;

    virtual QColor get_color(int x, int y) const   = 0;

    virtual QImage get_qImage() const              = 0;

    virtual void invert_color() {}

    virtual void grayscale() {}

    virtual void invert_color(int x1, int y1, int x2, int y2) {}

    virtual void grayscale(int x1, int y1, int x2, int y2) {}

    virtual void crop(int vertical_crop, int horizontal_crop,
Bmp::Resize_direction direction) {}

    virtual void expanse(int vertical_exp, int horizontal_crop,
Bmp::Resize_direction direction, QColor color) {}
};
```

1.4. Исключения

Все, перечисленные далее исключения являются членами пространства имен Bmp. Данные исключения были написаны для перехвата исключительных ситуаций, таких как: передача на открытие изображения с неправильным или

не поддерживающимся количеством битов на цвет; передача отрицательных аргументов для функций изменения размера изображения или создания нового.

1.4.1. Некорректный bitcount

```
class Bad_bitcount: public std::exception {
public:
    Bad_bitcount(int bitcount);
    const char* what() const throw();
    int err_bitcount() const;

private:
    int bitcount;
};
```

bitcount — количество битов на цвет, из-за которого данное исключения было брошено.

```
Bmp::Bad_bitcount::Bad_bitcount(int bitcount): std::exception{},
                                               bitcount(bitcount) {}

const char* Bmp::Bad_bitcount::what() const throw() {
    return "This version supports only 24 bitcount";
}

int Bmp::Bad_bitcount::err_bitcount() const {
    return bitcount;
}
```

В конструкторе переданный при бросании bitcount присваивается соответствующему полю, и далее конструктор делегирует дальнейшее создание базовому классу.

Метод what возвращает сообщение о том, что данная версия программы поддерживает только 24 бита на цвет.

Метод err_bitcount — геттер возвращающий значения не поддерживаемого bitcount.

1.4.2. Некорректный переданный размер при создании изображения

```
class Bad_size: public std::exception {
    public:
        Bad_size(int width, int height);
        const char* what() const throw();
        int err_width() const;
        int err_height() const;

    private:
        int width;
        int height;
};
```

width, heigth — переданный размер, из-за которого исключения было брошено.

```
Bmp::Bad_size::Bad_size(int width, int height): std::exception{},
                                                width(width), height(height) {}

const char* Bmp::Bad_size::what() const throw() {
    return "Incorrect image's raster size";
}

int Bmp::Bad_size::err_width() const {
    return width;
}

int Bmp::Bad_size::err_height() const {
    return height;
}
```

В конструкторе инициализируются поля данного исключения и далее создание объекта делегируется базовому классу.

Метод what возвращает сообщение о том, что был передан некорректный размер изображения.

Методы err_width, err_hegith — геттеры.

1.4.3. Некорректные параметры изменения размера изображения

```
class Bad_resize: public std::exception {
    public:
        Bad_resize(int vertical, int horizontal);
        const char* what() const throw();
        int err_vertical() const;
        int err_horizontal() const;

    private:
        int vertical;
        int horizontal;
    };
}
```

vertical, horizontal — параметры расширения или сжатия, вызвавшие исключение.

```
Bmp::Bad_resize::Bad_resize(int vertical, int horizontal): std::exception{},
                                                           vertical(vertical),
                                                           horizontal(horizontal) {}
const char* Bmp::Bad_resize::what() const throw() {
    return "Incorrect resize parameters";
}

int Bmp::Bad_resize::err_vertical() const {
    return vertical;
}

int Bmp::Bad_resize::err_horizontal() const {
    return horizontal;
}
```

В конструкторе инициализируются поля данного исключения и далее создание объекта делегируется базовому классу.

Метод what возвращает сообщение о том, что был передан некорректные параметры изменения размера.

Методы err_vertical, err_horizontal — геттеры.

2. BMP_IMAGE24

Класс для реализации требуемого функционала и представления изображения. Данный класс публично наследуется от класса bmp_image.

Поля данного класса (все приватные):

```
int size;
int height;
short bitcount;
uint8_t** raster;
int width;
```

Макрос для вычисления количества выравнивающих байт:

```
#define ALIGNMENT24(width) (4*(((width)*3)%4 ? 1 : 0) - ((width)*3)%4)
```

2.1. Конструкторы

2.1.1. Путь к изображению

```
Bmp_image24::Bmp_image24(std::string file_path) {
    if (file_path != "") {
        BitMapFileHeader bm_header;
        BitMapInfo       bm_info;

        std::ifstream file(file_path, std::ios::in | std::ios::binary);
        if (file.is_open()) {
            file.read(reinterpret_cast<char*>(&bm_header),
sizeof(BitMapFileHeader));
            file.read(reinterpret_cast<char*>(&bm_info), sizeof(BitMapInfo));

            width    = bm_info.biWidth;
            height   = bm_info.biHeight;
            size     = bm_info.biSizeImage;
            bitcount = bm_info.biBitCount;

            // allocation memory for raster image
            raster = new uint8_t*[height];
            raster[0] = new uint8_t[size];
            for (int i = 1; i != height; ++i) {
                raster[i] = raster[i-1] + ((width*3) + ALIGNMENT24(width));
            }

            file.seekg(bm_header.bfOffBits); // here raster starts
            for (int i = 0; i != height; ++i) {
                for (int j = 0; j != width*3 + ALIGNMENT24(width); ++j) {
                    file.read(reinterpret_cast<char*>(&raster[i][j]), 1);
                }
            }

            file.close();
        }
    }
}
```

После открытия файла в раннее объявленные структуры записываются структуры открываемого файла. Инициализируются поля класса, аллоцируется память (двумерный массив) под растровое изображения и далее побайтно заполняется.

2.1.2. Ширина, высота, растровое изображение

```
Bmp_image24::Bmp_image24(int width, int height, uint8_t* raster):
            size(height * ((width*3) + ALIGNMENT24(width))), // 24 - bitcount
                                                  width(width),
                                                  height(height),
                                                 bitcount(24) {
    if (width < 0 || height < 0) {
        throw Bmp::Bad_size(width, height);
    }

    this->raster = new uint8_t*[height];
    this->raster[0] = new uint8_t[size]{};
    for (int i = 1; i != height; ++i) {
        this->raster[i] = this->raster[i-1] + ((width*3) + ALIGNMENT24(width));
    }

    if (raster) {
        for (int i = 0; i != height; ++i) {
            for (int j = 0; j != (width*3); ++j) {
                this->raster[i][j] = raster[i*(width*3) + j];
            }
        }
    } else {
        for (int i = 0; i != height; ++i) {
            // null init for all bytes in new raster
            for (int j = 0; j != ((width*3) + ALIGNMENT24(width)); ++j) {
                this->raster[i][j] = 255;
            }
        }
    }
}
```

Инициализируются поля объекта

Если переданные ширина или высота некорректные, бросается исключение.

Аллоцируется память под растр, если был передан массив байт, то выделенная память заполняется значениями данного массива, если нет (случай по умолчанию), то весь растр просто заполняется белым цветом.

2.2. Конструкторы копирования

2.2.1. Конструктор копирования

```
Bmp_image24::Bmp_image24(Bmp_image24 const& other) {
    size     = other.size;
    width    = other.width;
    height   = other.height;
    bitcount = other.bitcount;

    // allocation
    raster = new uint8_t*[height];
    raster[0] = new uint8_t[size];
    for (int i = 1; i != height; ++i) {
        raster[i] = raster[i-1] + ((width*3) + ALIGNMENT24(width));
    }

    for (int i = 0; i != height; ++i) {
        for (int j = 0; j != (width*3) + ALIGNMENT24(width); ++j) {
            raster[i][j] = other.raster[i][j];
        }
    }
}
```

Копируемое изображение передается по ссылке, чтобы избежать излишнего копирования. Копируются поля. Аллоцируется память и также копируется растер.

2.2.2. Перемещающий конструктор копирования

```
Bmp_image24::Bmp_image24(Bmp_image24&& other): size(other.size),
                                               width(other.width),
                                               height(other.height),
                                               bitcount(other.bitcount) {
    other.size     = 0;
    other.width    = 0;
    other.height   = 0;
    other.bitcount = 0;

    // allocation
    raster = new uint8_t*[other.get_height()];
    raster[0] = new uint8_t[other.get_size()];
    for (int i = 0; i != height; ++i) {
        for (int j = 0; j != (width*3) + ALIGNMENT24(width); ++j) {
            raster[i][j] = other.raster[i][j];
        }
    }

    other.raster = nullptr;
}
```

Копируются поля копируемого объекта и зануляются. Далее копируется растер и также зануляется.

2.3 Операторы присваивания

2.3.1. Оператор присваивания

```
Bmp_image24& Bmp_image24::operator=(Bmp_image24 const& other) {
    delete [] raster;
    delete [] raster[0];

    size     = other.size;
    width    = other.width;
    height   = other.height;
    bitcount = other.bitcount;

    // allocation
    raster = new uint8_t*[other.get_height()];
    raster[0] = new uint8_t[other.get_size()];
    for (int i = 0; i != height; ++i) {
        for (int j = 0; j != (width*3) + ALIGNMENT24(width); ++j) {
            raster[i][j] = other.raster[i][j];
        }
    }

    return *this;
```

}Удаляется предыдущий растр и выделяется память по новый. Копируются поля, присваиваемого объекта и его растр.

2.3.2. Перемещающий оператор присваивания

```
Bmp_image24& Bmp_image24::operator=(Bmp_image24&& other) {
    delete [] raster[0];
    delete [] raster;

    size     = other.size;
    height   = other.height;
    width    = other.width;
    bitcount = other.bitcount;

    other.size     = 0;
    other.width    = 0;
    other.height   = 0;
    other.bitcount = 0;

    // allocation
    raster = new uint8_t*[other.get_height()];
    raster[0] = new uint8_t[other.get_size()];
    for (int i = 0; i != height; ++i) {
        for (int j = 0; j != (width*3) + ALIGNMENT24(width); ++j) {
            raster[i][j] = other.raster[i][j];
        }
    }

    other.raster = nullptr;
    return *this;
```

}Аналогично перемещающему конструктору копирования.

2.4. Деструктор

```
Bmp_image24::~Bmp_image24() {
    delete [] raster[0];
    delete [] raster;
}
```

Освобождается память, выделенная под растровое изображение.

2.5. Сеттеры

2.5.1. set_color

```
void Bmp_image24::set_color(int x, int y, QColor const& color) {
    raster[height-y-1][x*3]   = color.blue();
    raster[height-y-1][x*3+1] = color.green();
    raster[height-y-1][x*3+2] = color.red();
}
```

Функция получает координаты пикселя, цвет которого нужно поменять и сам цвет.
Индексы элементов, которым присваивается значения нового цвета становятся очевидными, если учесть то, что bmp-файле обратный порядок строк.

2.6. Геттеры

2.6.1 Базовые геттеры

```
int Bmp_image24::get_size() const {
      return size;
}

int Bmp_image24::get_height() const {
    return height;
}

int Bmp_image24::get_width() const {
    return width;
}

short Bmp_image24::get_bitcount() const {
    return bitcount;
}

uint8_t Bmp_image24::get_raster(int i, int j) const {
    return raster[i][j];
}
```

Метод get_raster — геттер, возвращающий значение байта на i-ой строке, j-ом столбце растра.

2.6.2. get_raster

```
uint8_t* Bmp_image24::get_raster() const {
    uint8_t* copy;
    copy = new uint8_t[height * width*3];
    for (int i = 0; i != height; ++i) {
        for (int j = 0; j != width*3; ++j) {
            copy[i*(width*3) + j] = raster[i][j];
        }
    }

    return copy;

}
```

Метод аллоцирует память, копирует в нее растр и возвращает.

2.6.3. get_color

```
QColor Bmp_image24::get_color(int x, int y) const {
    return qRgb(raster[height-y-1][x*3 + 2],
                raster[height-y-1][x*3 + 1],
                raster[height-y-1][x*3]);
}
```

Метод возвращает экземпляр класса QRgb.

2.6.4. get_qimage

```
QImage Bmp_image24::get_qImage() const {
    QImage image(width, height, QImage::Format_RGB888);
    for (int i = 0; i != height; ++i) {
        for (int j = width-1; j >= 0; --j) {
            image.setPixelColor(j, i, this->get_color(j, i));
        }
    }

    return image;
}
```

Создается экземпляр класса QImage с форматом, соответствующим 24-битному изображению. В цикле каждый пиксель image заполняется нужным цветом с помощью раннее описанного метода **get_color**.

2.7. Инверсия цвета

2.7.1. Инверсия всего изображения

```
void Bmp_image24::invert_color() {
    for (int i = 0; i != height; ++i) {
        for (int j = 0; j != (width*3) + ALIGNMENT24(width); ++j) {
            raster[i][j] = 255 - raster[i][j];
        }
    }
}
```

Каждый пиксель изображения побайтно инвертируется.

2.7.2. Инверсия ограниченной прямоугольной области

```
void Bmp_image24::invert_color(int x1, int y1, int x2, int y2) {
    if (x1 < 0) x1 = 0;
    if (y1 < 0) y1 = 0;
    if (x2 < 0) x2 = 0;
    if (y2 < 0) y2 = 0;

    int x_min = std::min(std::min(x1, x2), width);
    int y_min = std::min(std::min(y1, y2), height);
    int x_max = std::min(std::max(x1, x2), width);
    int y_max = std::min(std::max(y1, y2), height);

    for (int y = y_min; y != y_max; ++y) {
        for (int x = x_min; x != x_max; ++x) {
            this->set_color(x, y, QColor(255 - raster[height-y-1][x*3+2],
                                         255 - raster[height-y-1][x*3+1],
                                         255 - raster[height-y-1][x*3]));
        }
    }

}
```

Методу передаются координаты пикселей двух диагонально противоположных вершин прямоугольника. Если координаты вершины выходят за пределы изображения (координаты меньше нуля) то они зануляются. Далее, находятся минимальные и максимальные координаты путем сравнения минимальных координат, переданных точек и щирины или высоты изображения соответственно (чтобы избежать выход за пределы изображения).

Далее в цикле используется раннее описанный метод **set_color**, с его помощью каждый пиксель инвертируется, как было показано раннее.

2.8. Преобразование изображения в черно-белое

2.8.1. Преобразование всего изображения в черно-белое

```
void Bmp_image24::grayscale() {
    for (int i = 0; i != height; ++i) {
        for (int j = 0; j < (width*3) + ALIGNMENT24(width) - 3; j+=3) {
            uint8_t luma = 0.2126*raster[i][j] + 0.7152*raster[i][j+1] +
0.0722*raster[i][j+2];
            raster[i][j] = luma;
            raster[i][j+1] = luma;
            raster[i][j+2] = luma;
        }
    }
}
```

В цикле для каждого пикселя считается luma — относительная яркость цвета и присваивается каждому байту пикселя.

2.8.2. Преобразование ограниченной прямоугольной области изображения в черно-белое

```
void Bmp_image24::grayscale(int x1, int y1, int x2, int y2) {
    if (x1 < 0) x1 = 0;
    if (y1 < 0) y1 = 0;
    if (x2 < 0) x2 = 0;
    if (y2 < 0) y2 = 0;

    int x_min = std::min(std::min(x1, x2), width);
    int y_min = std::min(std::min(y1, y2), height);
    int x_max = std::min(std::max(x1, x2), width);
    int y_max = std::min(std::max(y1, y2), height);

    for (int y = y_min; y != y_max; ++y) {
        for (int x = x_min; x != x_max; ++x) {
            QColor rgb = get_color(x, y);
            uint8_t luma = 0.2126*rgb.red() + 0.7152*rgb.green() +
0.0722*rgb.blue();
            set_color(x, y, QColor(luma, luma, luma));
        }
    }
}
```

Вычисление координат, по которым будет пробегать цикл происходит аналогично методу инверсии цвета в области. В цикле на каждой итерации считывается цвет с помощью раннее описанного геттера, вычисляется luma и с помощью сеттера нужный пиксель заполняется цветом со всеми компонентами, равными luma.

2.9. Обрезание изображения

```
void Bmp_image24::crop(int vertical_crop, int horizontal_crop,
                       Bmp::Resize_direction direction) {
    if (vertical_crop < 0 || horizontal_crop < 0) {
        throw Bmp::Bad_resize{vertical_crop, horizontal_crop};
    }

    height -= vertical_crop;
    width -= horizontal_crop;
    size = height * (width*3 + ALIGNMENT24(width));

    uint8_t** cropped_raster = new uint8_t*[height];
    cropped_raster[0] = new uint8_t[size]{};
    for (int i = 1; i != height; ++i) {
        cropped_raster[i] = cropped_raster[i-1] + ((width*3) +
ALIGNMENT24(width));
    }
```

```
    switch (direction) {
        case Bmp::Resize_direction::center:
            vertical_crop /= 2;
            horizontal_crop /= 2;
            for (int i = 0; i != height; ++i) {
                for (int j = 0; j != width*3 + ALIGNMENT24(width); ++j) {
                    cropped_raster[i][j] =
raster[i+vertical_crop][j+horizontal_crop * 3];
                }
            } break;

        case Bmp::Resize_direction::upper_left:
            for (int i = 0; i != height; ++i) {
                for (int j = 0; j != width*3 + ALIGNMENT24(width); ++j) {
                    cropped_raster[i][j] = raster[i][j+horizontal_crop*3];
                }
            } break;

        case Bmp::Resize_direction::upper_right:
            for (int i = 0; i != height; ++i) {
                for (int j = 0; j != width*3 + ALIGNMENT24(width); ++j) {
                    cropped_raster[i][j] = raster[i][j];
                }
            } break;

        case Bmp::Resize_direction::lower_right:
            for (int i = 0; i != height; ++i) {
                for (int j = 0; j != width*3 + ALIGNMENT24(width); ++j) {
                    cropped_raster[i][j] = raster[i+vertical_crop][j];
                }
            } break;

        case Bmp::Resize_direction::lower_left:
            for (int i = 0; i != height; ++i) {
                for (int j = 0; j != width*3 + ALIGNMENT24(width); ++j) {
                    cropped_raster[i][j] =
raster[i+vertical_crop][j+horizontal_crop*3];
                }
            } break;
    }

    delete [] raster[0];
    delete [] raster;

    raster = cropped_raster;
    raster[0] = cropped_raster[0];
}
```

Метод принимает три аргумента: количество обрезаемых пикселей по горизонтали, вертикали и направление (или точка) относительно которого будет проводится обрезание. Если было передано некорректное кол-во пикселей, то бросается исключение **Bad_resize**. Изменяются соответствующие поля, выделяется память под новый растр. Затем в зависимости от направления выполняется нужный алгоритм.

2.10. Расширение изображения

```
void Bmp_image24::expanse(int vertical_exp, int horizontal_exp,
Bmp::Resize_direction direction, QColor color) {
    if (vertical_exp < 0 || horizontal_exp < 0) {
        throw Bmp::Bad_resize{vertical_exp, horizontal_exp};
    }

    int old_width = width;
    int old_height = height;

    height += vertical_exp;
    width += horizontal_exp;
    size = height * (width*3 + ALIGNMENT24(width));

    uint8_t** expansed_raster = new uint8_t*[height];
    expansed_raster[0] = new uint8_t[size]{};
    for (int i = 1; i != height; ++i) {
        expansed_raster[i] = expansed_raster[i-1] + ((width*3) +
ALIGNMENT24(width));
    }

    switch(direction) {
        case Bmp::Resize_direction::center:
            vertical_exp /= 2;
            horizontal_exp /= 2;
            for (int i = 0; i != vertical_exp; ++i) {
               for (int j = 0; j < (width*3) - 2; j += 3) {
                   expansed_raster[i][j] = color.blue();
                   expansed_raster[i][j+1] = color.green();
                   expansed_raster[i][j+2] = color.red();
               }
            }

            for (int i = vertical_exp; i != old_height + vertical_exp; ++i) {
                for (int j = 0; j < horizontal_exp*3 - 2; j += 3) {
                    expansed_raster[i][j] = color.blue();
                    expansed_raster[i][j+1] = color.green();
                    expansed_raster[i][j+2] = color.red();
                }

                for (int j = horizontal_exp*3; j != old_width*3 +
horizontal_exp*3; ++j) {
                    expansed_raster[i][j] = raster[i-vertical_exp][j-
horizontal_exp*3];
                }

                for (int j = old_width*3 + horizontal_exp*3; j < width*3 - 2; j
+= 3) {
                    expansed_raster[i][j] = color.blue();
                    expansed_raster[i][j+1] = color.green();
                    expansed_raster[i][j+2] = color.red();
                }
            }

            for (int i = old_height + vertical_exp; i != height; ++i) {
                for (int j = 0; j < (width*3) - 2; j += 3) {
                    expansed_raster[i][j] = color.blue();
                    expansed_raster[i][j+1] = color.green();
                    expansed_raster[i][j+2] = color.red();
```

```
                }
            } break;

        case Bmp::Resize_direction::upper_left:
            for (int i = 0; i != old_height; ++i) {
                for (int j = 0; j < horizontal_exp*3 - 2; j += 3) {
                    expansed_raster[i][j] = color.blue();
                    expansed_raster[i][j+1] = color.green();
                    expansed_raster[i][j+2] = color.red();
                }

                for (int j = horizontal_exp*3; j != (width*3) +
ALIGNMENT24(width); ++j) {
                    expansed_raster[i][j] = raster[i][j-horizontal_exp*3];
                }
            }
            for (int i = old_height; i != height; ++i) {
                for (int j = 0; j < width*3 - 2; j += 3) {
                    expansed_raster[i][j] = color.blue();
                    expansed_raster[i][j+1] = color.green();
                    expansed_raster[i][j+2] = color.red();
                }
            } break;

        case Bmp::Resize_direction::upper_right:
            for (int i = 0; i != old_height; ++i) {
                for (int j = 0; j != (old_width)*3; ++j) {
                    expansed_raster[i][j] = raster[i][j];
                }

                for (int j = old_width*3; j < width*3 - 2; j += 3) {
                    expansed_raster[i][j] = color.blue();
                    expansed_raster[i][j+1] = color.green();
                    expansed_raster[i][j+2] = color.red();
                }
            }

            for (int i = old_height; i != height; ++i) {
                for (int j = 0; j < width*3 - 2; j += 3) {
                    expansed_raster[i][j] = color.blue();
                    expansed_raster[i][j+1] = color.green();
                    expansed_raster[i][j+2] = color.red();
                }
            } break;

        case Bmp::Resize_direction::lower_right:
            for (int i = 0; i != height - old_height; ++i) {
                for (int j = 0; j < width*3 - 2; j += 3) {
                    expansed_raster[i][j] = color.blue();
                    expansed_raster[i][j+1] = color.green();
                    expansed_raster[i][j+2] = color.red();
                }
            }

            for (int i = height - old_height; i != height; ++i) {
                for (int j = 0; j != old_width*3; ++j) {
                    expansed_raster[i][j] = raster[i-vertical_exp][j];
                }

                for (int j = old_width*3; j < width*3 - 2; j += 3) {
                    expansed_raster[i][j] = color.blue();
```

```
                    expansed_raster[i][j+1] = color.green();
                    expansed_raster[i][j+2] = color.red();
                }
            } break;

        case Bmp::Resize_direction::lower_left:
            for (int i = 0; i != height - old_height; ++i) {
                for (int j = 0; j < (width*3) - 2; j += 3) {
                    expansed_raster[i][j] = color.blue();
                    expansed_raster[i][j+1] = color.green();
                    expansed_raster[i][j+2] = color.red();
                }
            }

            for (int i = height - old_height; i != height; ++i) {
                for (int j = 0; j < horizontal_exp*3 - 2; j += 3) {
                    expansed_raster[i][j] = color.blue();
                    expansed_raster[i][j+1] = color.green();
                    expansed_raster[i][j+2] = color.red();
                }

                for (int j = horizontal_exp*3; j != (width*3) +
ALIGNMENT24(width); ++j) {
                    expansed_raster[i][j] = raster[i-vertical_exp][j -
horizontal_exp*3];
                }
            } break;
    }

    delete [] raster[0];
    delete [] raster;

    raster = expansed_raster;
    raster[0] = expansed_raster[0];
}
```

Метод принимает четыре аргумента: количество добавляемых пикселей по горизонтали, вертикали, направление (или точка) относительно которого будет проводится расширение и цвет заливки фона. Если было передано некорректное кол-во пикселей, то бросается исключение **Bad_resize**. Изменяются соответствующие поля, выделяется память под новый растр. Затем в зависимости от направления выполняется нужный алгоритм, в каждом из которых пиксели фона заполняются нужными значениями, соответствующими значениям переданного цвета.

3. BMP

Данный namespace включает в себя функции для полиморфной работы с классом Bmp_image.

3.1. Конструктор для всех классов

```
Bmp_image* Bmp::bmp(std::string file_path) {
    if (file_path != "") {
        std::ifstream file(file_path, std::ios::in | std::ios::binary);
        if (file.is_open()) {
            char check[3]{};
            file.read(check, 2);

            if (std::string(check) == "BM") {
                short bitcount;
                file.seekg(BM_BITCOUNT_INDEX);
                file.read(reinterpret_cast<char*>(&bitcount), sizeof(short));
                file.close();

                Bmp_image* bmp_image = nullptr;
                switch(static_cast<int>(bitcount)) {
                    case 24:
                        try {
                            bmp_image = new Bmp_image24(file_path);
                        } catch (std::bad_alloc& e) {
                            QErrorMessage err_msg;
                            err_msg.setWindowTitle("Bad allocation");
                            err_msg.showMessage(QString("Not enough memory to do
this operation"));
                            err_msg.exec();

                            bmp_image = nullptr;
                        } break;
                    default:
                        throw Bmp::Bad_bitcount{static_cast<int>(bitcount)};
                }

                return bmp_image;
            }
        }
    }

    return nullptr;
}
```

Данная функция в зависимости от количества бит на цвет вызывает нужный конструктор, в случае ошибки выделения памяти пользователю показывается соответствующие окно с ошибкой. В случае не поддерживаемой версии bmp бросается исключение **Bmp::Bad_bitcount**.

3.2. КопированиеBmp_image* Bmp::copy(Bmp_image *image) {

```
КопированиеBmp_image* Bmp::copy(Bmp_image *image) {
    Bmp_image* copy;
    switch(image->get_bitcount()) {
        case 24: copy = new Bmp_image24(*static_cast<Bmp_image24*>(image));
break;
```

```
    }

    return copy;
}
```

Возвращает указатель на созданную, в зависимости от битности изображения, копию.

3.3. Создание нового изображения

```
Bmp_image* Bmp::create(int width, int height) {
    Bmp_image* image = new Bmp_image24(width, height);
    return image;
}
```

Возвращает указатель на созданное изображения с шириной **width** и высотой **height**.

3.4. Сохранение изображения

```
void Bmp::save(Bmp_image *image, std::string file_path) {
    std::ofstream file(file_path, std::ios::out | std::ios::binary);

    if (file.is_open()) {
        BitMapFileHeader bm_header;
        BitMapInfo       bm_info;

        bm_header.bfType1     = 'B';
        bm_header.bfType2     = 'M';
        bm_header.bfSize      = sizeof(BitMapFileHeader) +
                                sizeof(BitMapInfo) +
                                image->get_size();
        bm_header.bfReserved1 = bm_header.bfReserved2 = 0;

        bm_info.biSize          = BI_SIZE;
        bm_info.biWidth         = image->get_width();
        bm_info.biHeight        = image->get_height();
        bm_info.biPlanes        = 1;
        bm_info.biCompression   = 0;
        bm_info.biSizeImage     = image->get_size();
        bm_info.biXPelsPerMeter = 0;
        bm_info.biYPelsPerMeter = 0;

        switch (image->get_bitcount()) {
            case 24:
                bm_header.bfOffBits    = 54;
                bm_info.biBitCount     = 24;
                bm_info.biClrUsed      = 0;
                bm_info.biClrImportant = 0;

                file.write(reinterpret_cast<char*>(&bm_header),
sizeof(BitMapFileHeader));
                file.write(reinterpret_cast<char*>(&bm_info),
sizeof(BitMapInfo));

                uint8_t** saving_raster = new uint8_t*[bm_info.biHeight];
                saving_raster[0] = new uint8_t[bm_info.biSizeImage]{};
                for (int i = 1; i != bm_info.biHeight; ++i) {
```

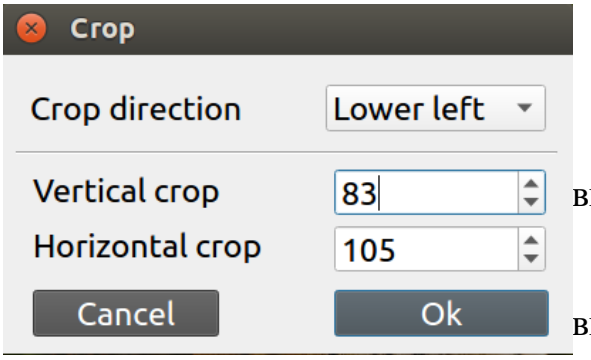
```
                    saving_raster[i] = saving_raster[i-1] + (bm_info.biWidth*3)
+ ALIGNMENT24(bm_info.biWidth);
                }

                for (int i = 0; i != bm_info.biHeight; ++i) {
                    for (int j = 0; j != bm_info.biWidth*3; ++j) {
                        saving_raster[i][j] = image->get_raster(i, j);
                    }
                }

                file.write(reinterpret_cast<char*>(saving_raster[0]),
                                                    bm_info.biHeight *
((bm_info.biWidth*3) + ALIGNMENT24(bm_info.biWidth)));
                file.close();

                delete [] saving_raster[0];
                delete [] saving_raster;
            break;
        }
    } else {
        QErrorMessage err_msg;
        err_msg.setWindowTitle("Saving file error");
        err_msg.showMessage("Can't save this file on some reason.");
        err_msg.setStyleSheet("QPushButton {"
                              "    color: white;"
                              "    background-color: rgb(75, 75, 75);"
                              "}");
        err_msg.exec();
    }
}
```

В случае удачного открытия файла на запись, создается две структуры для bmp изображения. Поля данных структур заполняются нужной информацией. Затем весь растр одни блоком записывается в файл.

4. QT КЛАССЫ

Все диалоговые окна созданы с помощью Qt. В каждом разделе присутствует рисунок, показывающий графическую составляющую, каждого из окон.

4.1. Диалоговое окно обрезания изображения



Интерфейс представлен на рисунке 1. Выбор направления обрезки выполняется засчет выпадающего списка. Количество обрезаемых пикселей выбирается с помощью spinbox.

4.1.1. Конструктор

```
Crop_dialog::Crop_dialog(int max_width, int max_height, QWidget *parent):
Qdialog(parent), ui(new Ui::Crop_dialog) {
    ui->setupUi(this);
    ui->vertical_crop->setMaximum(max_height-1);
    ui->horizontal_crop->setMaximum(max_width-1);
    setWindowTitle("Crop");

    read_settings();

}
```

Конструктор принимает максимальное значения обрезание картинки по ширине и высоте. Устанавливается интерфейс, и максимальное значения для spinbox`ов. Устанавливается название окна и считываются настройки методом **read_settings**, который будет описан позже.

4.1.2. Настройки

```
void Crop_dialog::write_settings() {
    QSettings settings("My Soft", "LULpainter");

    settings.beginGroup("Crop dialog");
    settings.setValue("vertical_crop", ui->vertical_crop->value());
    settings.setValue("horizontal_crop", ui->horizontal_crop->value());
    settings.setValue("crop_direction", ui->crop_direction->currentIndex());
    settings.setValue("size", size());
    settings.setValue("pos", pos());
    settings.endGroup();
}

void Crop_dialog::read_settings() {
    QSettings settings("My Soft", "LULpainter");

    settings.beginGroup("Crop dialog");
    ui->vertical_crop->setValue(settings.value("vertical_crop", 0).toInt());
    ui->horizontal_crop->setValue(settings.value("horizontal_crop", 0).toInt());
    ui->crop_direction->setCurrentIndex(settings.value("crop_direction",
0).toInt());
```

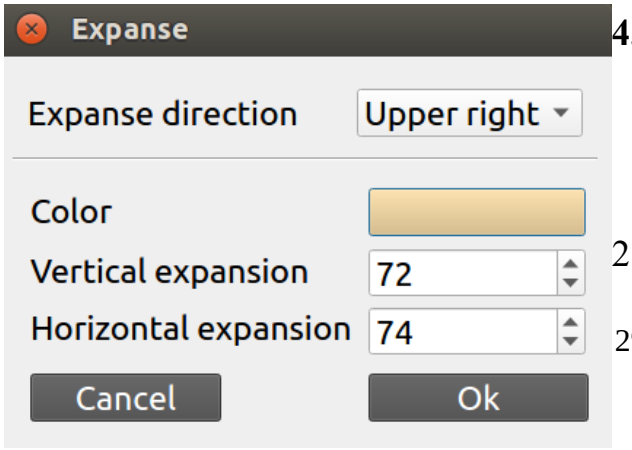
```
    resize(settings.value("size", QSize(403, 211)).toSize());
    move(settings.value("pos", QPoint(200, 200)).toPoint());
    settings.endGroup();
}
```

В методе **write_settings** записывается информация о положения окна на момент закрытия, его размер, и значения заполняемых полей. В методе **read_settings** все эти настройки считываются.

4.1.3. Обработка нажатий кнопок

```
void Crop_dialog::on_cancel_button_clicked() {
    write_settings();
    close();
}

void Crop_dialog::on_ok_button_clicked() {
    Bmp::Resize_direction crop_direction;
    switch(ui->crop_direction->currentIndex()) {
        case 0: crop_direction = Bmp::Resize_direction::center; break;
        case 1: crop_direction = Bmp::Resize_direction::upper_left; break;
        case 2: crop_direction = Bmp::Resize_direction::upper_right; break;
        case 3: crop_direction = Bmp::Resize_direction::lower_right; break;
        case 4: crop_direction = Bmp::Resize_direction::lower_left; break;
    }

    emit ok_button_clicked(ui->vertical_crop->value(),
                           ui->horizontal_crop->value(),
                           crop_direction);

    write_settings();
    close();
}
```

В случае нажатия кнопки **cancel** просто записываются настройки и окно закрывается. Если была нажата кнопка **ok**, то испускается сигнал с аргументами: размер обрезания по вертикали, по горизонтали и направление обрезки. (далее в **MainWindow** данный сигнал будет обработан).



4.2. Диалоговое окно расширения изображения

Интерфейс представлен на рисунке 2.Выбор направления обрезки

выполняется засчет выпадающего списка.

Количество обрезаемых пикселей выбирается с помощью spinbox.

Цвет выбирается с помощью QColorDialog.

4.2.1. Конструктор

```
Expanse_dialog::Expanse_dialog(QWidget *parent): QDialog(parent),
                                                 ui(new Ui::Expanse_dialog) {
    ui->setupUi(this);
    setWindowTitle("Expanse");

    read_settings();
}
```

Устанавливается интерфейс, название окна и считываются настройки методом **read_settings**, который будет описан позже.

4.2.2 Установка цвета кнопки выбора цвета

```
void Expanse_dialog::set_background_color(QColor color) {
    ui->color_button->setStyleSheet(QString("background: rgb(") +
QString::number(color.red()) + QString(", ")
                                                               +
QString::number(color.green()) + QString(", ")
                                                               +
QString::number(color.blue()) + QString(");"));

}
```

Данный метод закрашивает кнопку выбора цвета в выбранный последним цвет с использованием CSS.

4.2.3. Настройки

```
void Expanse_dialog::write_settings() {
    QSettings settings("My Soft", "LULpainter");

    settings.beginGroup("Expanse dialog");
    settings.setValue("red", color.red());
    settings.setValue("green", color.green());
    settings.setValue("blue", color.blue());
    settings.setValue("vertical_crop", ui->vertical_crop->value());
    settings.setValue("horizontal_crop", ui->horizontal_crop->value());
    settings.setValue("expanse_direction", ui->crop_direction->currentIndex());
    settings.setValue("size", size());
    settings.setValue("pos", pos());
    settings.endGroup();
}

void Expanse_dialog::read_settings() {
    QSettings settings("My Soft", "LULpainter");
```

```
    settings.beginGroup("Expanse dialog");
    color.setRgb(qRgb(settings.value("red", 0).toInt(),
                      settings.value("green", 0).toInt(),
                      settings.value("blue", 0).toInt()));
    ui->vertical_crop->setValue(settings.value("vertical_crop", 0).toInt());
    ui->horizontal_crop->setValue(settings.value("horizontal_crop", 0).toInt());
    ui->crop_direction->setCurrentIndex(settings.value("expanse_direction",
0).toInt());
    set_background_color(color);
    resize(settings.value("size", QSize(403, 211)).toSize());
    move(settings.value("pos", QPoint(200, 200)).toPoint());
    settings.endGroup();
}
```

В методе **write_settings** записывается информация о положения окна на момент закрытия, его размер, значения заполняемых полей и значения red, green, blue выбранного последним цвета. В методе **read_settings** все эти настройки считываются.

4.2.4. Обработка нажатий кнопок

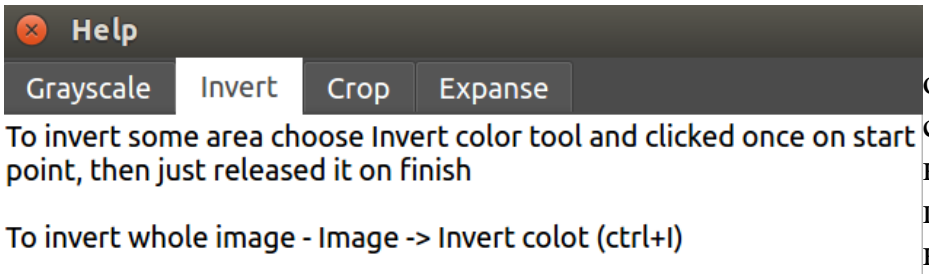
```
void Expanse_dialog::on_color_button_clicked() {
    color = QColorDialog::getColor();
    set_background_color(color);
}

void Expanse_dialog::on_cancel_button_clicked() {
    write_settings();
    close();
}

void Expanse_dialog::on_ok_button_clicked() {
    Bmp::Resize_direction expanse_direction;
    switch(ui->crop_direction->currentIndex()) {
        case 0: expanse_direction = Bmp::Resize_direction::center; break;
        case 1: expanse_direction = Bmp::Resize_direction::upper_left; break;
        case 2: expanse_direction = Bmp::Resize_direction::upper_right; break;
        case 3: expanse_direction = Bmp::Resize_direction::lower_right; break;
        case 4: expanse_direction = Bmp::Resize_direction::lower_left; break;
    }

    emit ok_button_clicked(ui->vertical_crop->value(),
                           ui->horizontal_crop->value(),
                           expanse_direction, color);

    write_settings();
    close();
}
```

При нажатии на кнопку выбора цвета, открывается диалоговое окно QColorDialog, пользователь выбирает цвет и с помощью метода **set_background_color** кнопка перекрашивается в выбранный цвет.

В случае нажатия кнопки **cancel** просто записываются настройки и окно закрывается. Если была нажата кнопка **ok**, то испускается сигнал с аргументами: размер обрезания по вертикали, по горизонтали, направление

обрезки и цвет заливки фона. (далее в **MainWindow** данный сигнал будет обработан).

4.3. Диалоговое окно „помощь“



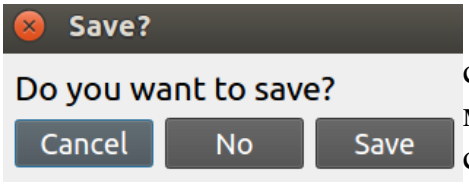
Данное диалоговое окно представляет из себя 4 вкладки с коротким описанием принципов работы с каждым из реализованных элементов функционала. Представленные вкладки: преобразование изображение в черно-белое, инвертирование цвета, обрезка, расишрение.

4.3.1. Конструктор

```
Help_dialog::Help_dialog(QWidget *parent): QDialog(parent),
                                           ui(new Ui::Help_dialog) {
    ui->setupUi(this);
    setWindowTitle("Help");
}
```

В конструкторе устанавливается пользовательский интерфейс и название диалогового окна „Help“.

4.4. Диалоговое окно сохранения



Данное диалоговое окно возникает в тех случаях, когда пользователь предположительно мог забыть сохранить проделанные действия и не сохранил их, на пример, выход из программы, открытие нового файла, создание нового файла.

4.4.1. Конструктор

```
Save_dialog::Save_dialog(QWidget *parent): QDialog(parent),
                                           ui(new Ui::save_quit_dialog) {
    ui->setupUi(this);
    this->setModal(true);
}
```

В конструкторе устанавливается пользовательский интерфейс и данное окно делается модальным.

4.4.2. Обработка нажатий кнопок

```
void Save_dialog::on_save_button_clicked() {
    emit save_button_clicked(this);
    done(0);
```

```
}

void Save_dialog::on_cancel_button_clicked() {
    emit cancel_button_clicked(true);
    done(0);
}

void Save_dialog::on_no_button_clicked() {
    done(0);
}

void Save_dialog::closeEvent(QCloseEvent *event) {
    emit cancel_button_clicked(true);
    event->accept();
}
```

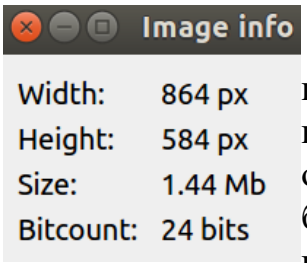
При нажатии на кнопку **Save** испускается сигнал **save_button_clicked(this)**.

При нажатии на кнопку **Cancel** испускается сигнал **cancel_button_clicked(true)**.

При нажатии на кнопку **No** испускается сигнал **no_button_clicked(this)**.

При закрывании окна исполняется **closeEvent**: испускается сигнал **cancel_button_clicked(true)**.

4.5. Диалоговое окно с информацией об изображении



Данное диалоговое окно содержит информацию об изображении: ширина в пикселях, высота в пикселях, размер непосредственно изображения и его битность.Данные в этом окне меняются динамически, то есть, если изображение будет, на пример, обрезано, информация динамически изменится.

4.5.1. Конструктор

```
Info_dialog::Info_dialog(Bmp_image const* image) {
    setWindowTitle("Image info");

    QHBoxLayout* content = new QHBoxLayout(this);
    QVBoxLayout* labels = new QVBoxLayout(this);
    QVBoxLayout* values = new QVBoxLayout(this);

    QLabel* width = new QLabel("Width:       ", this);
    QLabel* height = new QLabel("Height:      ", this);
    QLabel* size = new QLabel("Size:          ", this);
    QLabel* bitcount = new QLabel("Bitcount: ", this);
    labels->addWidget(width);
    labels->addWidget(height);
    labels->addWidget(size);
    labels->addWidget(bitcount);

    this->width = new QLabel(QString::number(image->get_width()) + QString("
px"), this);
```

```
    this->height = new QLabel(QString::number(image->get_height()) + QString("
px"), this);
    this->size = new QLabel(Info_dialog::make_beauty(image->get_size()), this);
    this->bitcount = new QLabel(QString::number(image->get_bitcount()) +
QString(" bits"), this);
    values->addWidget(this->width);
    values->addWidget(this->height);
    values->addWidget(this->size);
    values->addWidget(this->bitcount);

    content->addLayout(labels);
    content->addLayout(values);

    setLayout(content);
}
```

Устанавливается название окна „Image info“. Инициализируются переменные, отвечающие за слои, содержащие названия свойств, сами свойства и слой контейнер для двух данных. Записываются названия свойств, а затем значения. В строке, где устанавливается значение для размера используется данная функция **make_beauty**:

```
QString Info_dialog::make_beauty(int size) {
    std::vector<QString> units{" B", " Kb", " Mb", " Gb"};

    int power = round(log(size)/log(1024));
    power = std::min(power, static_cast<int>(units.size() - 1));
    double beauty_size = size / pow(1024, power);

    QString unit = units.at(power);
    return QString::number(beauty_size, 'g', 3) + unit;
}
```

Данная функция применяется для того, чтобы сделать размер более human-readeable.

4.5.2. Изменение информации

Для динамического изменения используется публичный слот данного диалогового окна **change_info:**

```
void Info_dialog::change_info(Bmp_image const* image) {
    width->setText(QString::number(image->get_width()) + QString(" px"));
    height->setText(QString::number(image->get_height()) + QString(" px"));
    size->setText(make_beauty(image->get_size()));
    bitcount->setText(QString::number(image->get_bitcount()) + QString("
bits"));
}
```

Функция принимает указатель на изображение, в соответствии с которым нужно изменить информацию, находящуюся в диалоговом окне. Каждое значение просто перезаписывается.

4.6. Класс, наследуемый от QGraphicsScene

Решение написать отдельный класс для сцены было продиктовано переопределением слотов стандартной сцены на требуемые для реализации программы.

Конструктор данного класса просто делегирует создание объекта базовому классу.

Класс содержит в себе два поля:

```
QPoint first;
QPoint last;
```

first — поле, отвечающее за координаты начала выделения области на данной сцене.

last — поле, отвечающее за координаты конца выделения области на данной сцене.

Для данных полей в классе есть стандартные геттеры **get_first**, **get_last** просто возвращающие копию на значения **first** и **last**.

4.6.1. Переопределенные методы

```
void My_graphics_scene::mouseMoveEvent(QGraphicsSceneMouseEvent *event) {
    last = event->scenePos().toPoint();
    emit mouseMoved();
}

void My_graphics_scene::mousePressEvent(QGraphicsSceneMouseEvent *event) {
    first = event->scenePos().toPoint();
    emit mousePressed();
}

void My_graphics_scene::mouseReleaseEvent(QGraphicsSceneMouseEvent *event) {
    last = event->scenePos().toPoint();
    emit mouseReleased();
}
```

При нажатии на кнопку мыши будет вызываться событие **mousePressEvent**. Данное событие характеризуется как начало выделения. Поле **first** — заполнятеся координатами нажатия кнопки мыши.

При отпускании кнопки поле **last** — заполняется координатами „конца“.

Также динамически все время изменяется значение последней точки выделения с помощью слота **mouseMoveEvent**.

5. MAIN WINDOW

Рисунок 6

Данный класс хранит в себе реализацию связей между пользователем и бизнес логикой.

В поля данного класс входят:

```
    My_graphics_scene*
scene;      Bmp_image* bmp_image;

;      Bmp_image* bmp_image;

    QString open_file_path;
    QString prev_file_path;

    bool grayscale_clicked;
```

```
    bool invert_clicked;
    bool cancel_clicked;
    bool changed;
```

scene — сцена, на которой будет располагаться изображение

bmp_image — изображение

open_file_path, **prev_file_path** — путь до открытого файла и до предыдущего соответственно.

Флаги:

grayscale_clicked — сейчас нажата кнопка преобразования изображения в черно белое.

invert_clicked — сейчас нажата кнопка инверсии цвета изображения

cancel_clicked — была нажата кнопка „**Cancel**“ в очередном возникшем диалоговом окне.

changed — флаг, отвечающий за то, что изображение было изменено.

5.1. Констркутор

```
MainWindow::MainWindow(QWidget *parent): QMainWindow(parent),
                                         ui(new Ui::MainWindow) {
    ui->setupUi(this);
    setCentralWidget(ui->graphics_view);

    scene = new My_graphics_scene();
    connect(scene, SIGNAL(mouseReleased()), this, SLOT(mouseReleased()));

    prev_file_path = "";
    bmp_image = nullptr;
    grayscale_clicked = false;
    invert_clicked = false;
    cancel_clicked = false;
    changed = false;

    ui->mainToolBar->widgetForAction(ui->actionCrop_3)            -
>setStyleSheet("width: 115px;");
    ui->mainToolBar->widgetForAction(ui->actioncoordinates_gray)  -
>setStyleSheet("width: 115px;");
    ui->mainToolBar->widgetForAction(ui->actioncoordinates_invert)-
>setStyleSheet("width: 115px;");
    ui->mainToolBar->widgetForAction(ui->actionExpanse)           -
>setStyleSheet("width: 115px;");

    read_settings();
}
```

QGraphicsView — устанавливается, как центральный виджет. Создается экземпляр, описанной раннее сцены. И соединяет сигнал, испускаемый сценой при отпускании мыши и слот **mouseReleased** главного окна, который обрабатывает данный сигнал.

5.1.1. mouseReleased

```
void MainWindow::mouseReleased() {
    int x1 = scene->get_first().x();
    int y1 = scene->get_first().y();
    int y2 = scene->get_last().y();
    int x2 = scene->get_last().x();

    if (grayscale_clicked) {
        bmp_image->grayscale(x1, y1, x2, y2);
        changed = true;
    } else if (invert_clicked) {
        bmp_image->invert_color(x1, y1, x2, y2);
        changed = true;
    }

    scene->clear();
    scene->addPixmap(QPixmap::fromImage(bmp_image->get_qImage()));
    ui->graphics_view->setScene(scene);
}
```

Сначала в переменные записываются начальные и конечные координаты выделенной области, затем в зависимости от выбранного инструмента вызывается соответствующая функция. Затем сцена очищается от старого не модифицированного изображения и на ней рисуется новое, модифицированное.

Теперь устанавливаются значения по умолчанию для полей класса и применяются стили для кнопок на панели инструментов.

С помощью метода **read_settings** считываются настройки

5.2. Настройки

5.2.1. Запись настроек

```
void MainWindow::write_settings() {
    QSettings settings("My Soft", "LULpainter");
```

```
    settings.beginGroup("Size and position");
    settings.setValue("size", size());
    settings.setValue("pos", pos());
    settings.endGroup();

    settings.setValue("file_path", open_file_path);
```

}В настройки записываются размер, положение главного окна и путь к открытому на данный момент изображению.

5.2.2. Считывание настроек

```
void MainWindow::read_settings() {
    QSettings settings("My Soft", "LULpainter");
    settings.beginGroup("Size and position");
    resize(settings.value("size", QSize(400, 400)).toSize());
    move(settings.value("pos", QPoint(200, 200)).toPoint());
    settings.endGroup();

    QString file_path = QString(settings.value("file_path", "").toString());
    if (file_path != "") {
        bmp_image = Bmp::bmp(file_path.toStdString());

        if (bmp_image) {
            open_file_path = file_path;
            this->setWindowTitle(QString("LULpainter ") + open_file_path);

            scene->addPixmap(QPixmap::fromImage(bmp_image->get_qImage()));
            ui->graphics_view->setScene(scene);
        }
    }
}
```

Устанавливается размер и положение главного окна, записанные в настройках. Далее, если в настройках присутствует запись о файле, то данное изображения открывается и помещается на сцену.

5.3. Деструктор

```
MainWindow::~MainWindow() {
    delete bmp_image;
    delete scene;
    delete ui;
}
```

Освобождается память под изображение и сцену.

```
5.4. Переключатели флагов
bool MainWindow::cancel_toggle() {
    return cancel_clicked = true;
}

void MainWindow::grayscale_toggle() {
```

```
    ui->mainToolBar->widgetForAction(ui->actioncoordinates_gray)-
>setStyleSheet("width: 115px;"

"background: rgb(150, 150, 150);");
    ui->mainToolBar->widgetForAction(ui->actioncoordinates_invert)-
>setStyleSheet("width: 115px;"

"background: rgb(75, 75, 75);");
    invert_clicked = false;
    grayscale_clicked = true;
}

void MainWindow::invert_toggle() {
    ui->mainToolBar->widgetForAction(ui->actioncoordinates_gray)-
>setStyleSheet("width: 115px;"

"background: rgb(75, 75, 75);");
    ui->mainToolBar->widgetForAction(ui->actioncoordinates_invert)-
>setStyleSheet("width: 115px;"

"background: rgb(150, 150, 150);");
    invert_clicked = true;
    grayscale_clicked = false;
}
```

cancel_toggle — слот, который нужен, чтобы связать сигнал, соответствующий нажатию кнопки **cancel**, с изменением флага **cancel_clicked**.

grayscale_toggle, **invert_toggle** — сначала меняются стили, чтобы визуально было понятно какой инструмент на данный момент выбран. Затем значения всех флагов инструментов, кроме выбранного зануляются.

5.5. Слот для Save As…

```
void MainWindow::save_as(QWidget *parent) {
    QString file_path = "";
    if (parent)
        file_path = QFileDialog::getSaveFileName(parent, "Save a file"); // in
case if file dialog will be exec from save_dialog
    else
        file_path = QFileDialog::getSaveFileName(this, "Save a file");

    if (file_path != "") {
        Bmp::save(bmp_image, file_path.toStdString());
        prev_file_path = open_file_path;
        open_file_path = file_path;

        Bmp_image* opening_bmp_image = Bmp::bmp(file_path.toStdString());
        if (opening_bmp_image) {
            bmp_image = opening_bmp_image;
            prev_file_path = open_file_path;
            open_file_path = file_path;
            this->setWindowTitle(QString("LULpainter ") + open_file_path);

            scene->clear();
            delete scene;
            scene = new My_graphics_scene;
            connect(scene, SIGNAL(mouseReleased()), this,
SLOT(mouseReleased()));
```

```
            scene->addPixmap(QPixmap::fromImage(bmp_image->get_qImage()));
            ui->graphics_view->setScene(scene);

            changed = false;
        } else {
            QErrorMessage err_msg;
            err_msg.setWindowTitle("Opening file error");
            err_msg.showMessage("Saving was successfull!"
                                "Can't open this file on some reason.");
            err_msg.setStyleSheet("QPushButton {"
                                  "    color: white;"
                                  "    background-color: rgb(75, 75, 75);"
                                  "}");
            err_msg.exec();
        }
    }

}
```

Функция принимает указатель на родительское окно, чтобы всегда возникать поверх него. Изображение сохраняется по новому пути, а затем открывается по нему. Открытие изображение анлогично тому, как оно открывается в конструкторе.

5.6. Crop_image

```
void MainWindow::crop_image(int vertical_crop, int horizontal_crop,
Bmp::Resize_direction crop_direction) {
    bmp_image->crop(vertical_crop, horizontal_crop, crop_direction);

    delete scene;
    scene = new My_graphics_scene;
    connect(scene, SIGNAL(mouseReleased()), this, SLOT(mouseReleased()));
    scene->addPixmap(QPixmap::fromImage(bmp_image->get_qImage()));

    ui->graphics_view->setScene(scene);

    emit image_changed(bmp_image);
    changed = true;

}
```

Слот нужен для перехвата сигнала, который испускается при нажатии на кнопку **ok** в **Crop_dialog**.

5.7. Expanse_image

```
 void MainWindow::expanse_image(int vertical_exp, int horizontal_exp,
                                Bmp::Resize_direction expanse_direction, QColor
color) {
    bmp_image->expanse(vertical_exp, horizontal_exp, expanse_direction, color);

    delete scene;
    scene = new My_graphics_scene;
    connect(scene, SIGNAL(mouseReleased()), this, SLOT(mouseReleased()));
    scene->addPixmap(QPixmap::fromImage(bmp_image->get_qImage()));
```

```
    ui->graphics_view->setScene(scene);

    emit image_changed(bmp_image);
    changed = true;

}
```

Слот нужен для перехвата сигнала, который испускается при нажатии на кнопку **ok** в **Expanse_dialog**.

5.8. on_actionOpen_triggered

```
void MainWindow::on_actionOpen_triggered() {
    QString file_path = QFileDialog::getOpenFileName(this, "Open a file");
    if (!file_path.isEmpty()) {
        if (changed) {
            Save_dialog* dialog = new Save_dialog(this);
            connect(dialog, SIGNAL(save_button_clicked()), this,
SLOT(on_actionSave_triggered()));
            connect(dialog, SIGNAL(cancel_button_clicked(bool)), this,
SLOT(cancel_toggle()));
            dialog->exec();
            delete dialog;

            if (cancel_clicked) {
                cancel_clicked = false;
                return;
            }
        }

        try {
            Bmp_image* opening_bmp_image = Bmp::bmp(file_path.toStdString());
            if (opening_bmp_image) {
                bmp_image = opening_bmp_image;
                prev_file_path = open_file_path;
                open_file_path = file_path;
                setWindowTitle(QString("LULpainter ") + open_file_path);

                scene->clear();
                delete scene;
                scene = new My_graphics_scene;
                connect(scene, SIGNAL(mouseReleased()), this,
SLOT(mouseReleased()));                                    :(
                scene->addPixmap(QPixmap::fromImage(bmp_image->get_qImage()));
                ui->graphics_view->setScene(scene);

                changed = false;
            } else {
                QErrorMessage err_msg;
                err_msg.setWindowTitle("Opening file error");
                err_msg.showMessage("Can't open this file on some reason.");
                err_msg.setStyleSheet("QPushButton {"
                                      "    color: white;"
                                      "    background-color: rgb(75, 75, 75);"
                                      "}");
                err_msg.exec();
            }
        } catch (Bmp::Bad_bitcount& err) {
            QErrorMessage err_box;
```

```
            err_box.setWindowTitle("Unsupported bitcount");
            err_box.showMessage(QString(err.what()) + QString("\nUploading
image's bitcount: ") + QString::number(err.err_bitcount()));
            err_box.setMaximumSize(490, 170);
            err_box.setMinimumSize(490, 170);
            err_box.setStyleSheet("QPushButton {"
                                  "    color: white;"
                                  "    background-color: rgb(75, 75, 75);"
                                  "}");
            err_box.exec();
        }
    }

}
```

В случае, если изображение было изменено, то открывается диалоговое окно с предложением сохранить изменения. Открывается изображение по переданному пути. Обновляются поля, устанавливается новое название главного окна. На новую сцену помещается новое изображение. В случае, если произошла ошибка или не удалось открыть файл пользователю будет выведена соответствующее окно с ошибкой.

5.9. Сохранение

```
void MainWindow::on_actionSave_triggered() {
    if (changed) {
        Bmp::save(bmp_image, open_file_path.toStdString());
        changed = false;
    }
}

void MainWindow::on_actionSave_As_triggered() {
    save_as(this);

}
```

В случае обычного сохранения, изображение просто сохраняется и флаг **changed** устанавливается в значение **false**. В случае сохранения как …, вызывается раннее описанный слот **save_as**.

5.10. Выбор инструментов

```
void MainWindow::on_actioncoordinates_gray_triggered() {
    grayscale_toggle();
}

void MainWindow::on_actioncoordinates_invert_triggered() {
    invert_toggle();
}
```

Вызываются описанные раннее слоты переключения инстурментов.

5.11. Применения инструмента ко всему изображению

```
void MainWindow::on_actionGraysacle_triggered() {
    bmp_image->grayscale();
    scene->clear();
    scene->addPixmap(QPixmap::fromImage(bmp_image->get_qImage()));
    ui->graphics_view->setScene(scene);

    changed = true;
}

void MainWindow::on_actionInvert_triggered() {
    bmp_image->invert_color();
    scene->clear();
    scene->addPixmap(QPixmap::fromImage(bmp_image->get_qImage()));
    ui->graphics_view->setScene(scene);

    changed = true;

}
```

Методы **bmp_image** применяется ко всему изображению.

5.12. Инструменты: Crop_dialog, Expanse_dialog

```
void MainWindow::on_actionCrop_3_triggered() {
    Crop_dialog* crop_dialog = new Crop_dialog(bmp_image->get_width(),
bmp_image->get_height(), this);
    connect(crop_dialog, SIGNAL(ok_button_clicked(int, int,
Bmp::Resize_direction)),
            this,        SLOT(crop_image(int, int, Bmp::Resize_direction)));

    crop_dialog->exec();
    delete crop_dialog;
}

void MainWindow::on_actionExpanse_triggered() {
    Expanse_dialog* expanse_dialog = new Expanse_dialog(this);
    connect(expanse_dialog, SIGNAL(ok_button_clicked(int, int,
Bmp::Resize_direction, QColor)),
            this,        SLOT(expanse_image(int, int, Bmp::Resize_direction,
QColor)));

    expanse_dialog->exec();
    delete expanse_dialog;

}
```

Создается диалоговое окно и соединяется с соответствующими слотами главного окна.

5.13. Окна-справки

```
void MainWindow::on_actionImage_info_triggered() {

    Info_dialog* info_dialog = new Info_dialog(bmp_image);
    connect(this, SIGNAL(image_changed(Bmp_image const*)), info_dialog,
SLOT(change_info(Bmp_image const*)));
```

```
    info_dialog->setAttribute(Qt::WA_DeleteOnClose);
    info_dialog->setWindowFlags(Qt::WindowStaysOnTopHint);
    info_dialog->show();
}

void MainWindow::on_actionHelp_triggered() {
    Help_dialog* help = new Help_dialog;
    help->setAttribute(Qt::WA_DeleteOnClose);
    help->show();
}
```

Создаются соответствующие диалоговые окна. В случае информационного окна сигнал главного об изменении изображения соединяется с раннее описанным слотом информационного.

5.14. Выход из приложения

```
void MainWindow::on_actionQuit_triggered() {
    close();

}
```

Вызывается метод **close** главного окна.

5.15. Close event

```
void MainWindow::closeEvent(QCloseEvent *event) {
    if (changed) {
        Save_dialog* dialog = new Save_dialog(this);

        if (open_file_path != "")
            connect(dialog, SIGNAL(save_button_clicked(QWidget*)), this,
SLOT(on_actionSave_triggered()));
        else
            connect(dialog, SIGNAL(save_button_clicked(QWidget*)), this,
SLOT(save_as(QWidget*)));

        connect(dialog, SIGNAL(cancel_button_clicked(bool)), this,
SLOT(cancel_toggle()));
        dialog->exec();
        delete dialog;
    }

    if (cancel_clicked) {
        event->ignore();
        cancel_clicked = false;
    } else {
        if (open_file_path == "")
            open_file_path = prev_file_path;

        write_settings();
        event->accept();
    }

}
```

В случае, если изображение было изменено пользователю предлагается сохранить проделанную работу. Создается диалоговое окно сохранения, соединяются все сигналы данного окна со слотами главного. Затем, в случае, если был испущен сигнал нажатия на кнопку **cancel**, **closeEvent** игнорируется. Если **cancel** не была нажата, то записываются настройки и приложение закрывается.

*В случае если, никакое изображение еще не открывалось, записывается путь к предыдущему изображению. Далее стане понятнее

5.16. Новое изображение

```
void MainWindow::on_actionNew_triggered() {
    if (changed) {
        Save_dialog* dialog = new Save_dialog(this);
        connect(dialog, SIGNAL(save_button_clicked(QWidget*)), this,
SLOT(on_actionSave_triggered()));
        connect(dialog, SIGNAL(cancel_button_clicked(bool)), this,
SLOT(cancel_toggle()));
        dialog->exec();
        delete dialog;
    }

    if (cancel_clicked) {
        cancel_clicked = false;
        return;
    }

    scene->clear();

    Bmp_image* opening_bmp_image = Bmp::create(ui->graphics_view->width(), ui-
>graphics_view->height());

    if (opening_bmp_image) {
        delete bmp_image;
        bmp_image = opening_bmp_image;
        prev_file_path = open_file_path;
        open_file_path = "";
        this->setWindowTitle(QString("LULpainter"));

        scene->addPixmap(QPixmap::fromImage(bmp_image->get_qImage()));
        ui->graphics_view->setScene(scene);

        changed = true;
    }


}
```

Сцена очищается перед тем, как поместить на нее новое изображение. Затем создается новое изображение с шириной и высотой, как у панели, в которой располагается изображение.

ЗАКЛЮЧЕНИЕ

В данной работе были созданы функции, для считывания и записи bmp файла. Реализованы требуемые в задании элементы функционала, такие как:

1. Инверсия цвета в заданной прямоугольной области;

2. Преобразование в черно-белое изображение в заданной прямоугольной области;

3. Изменение размера изображения путем обрезки.

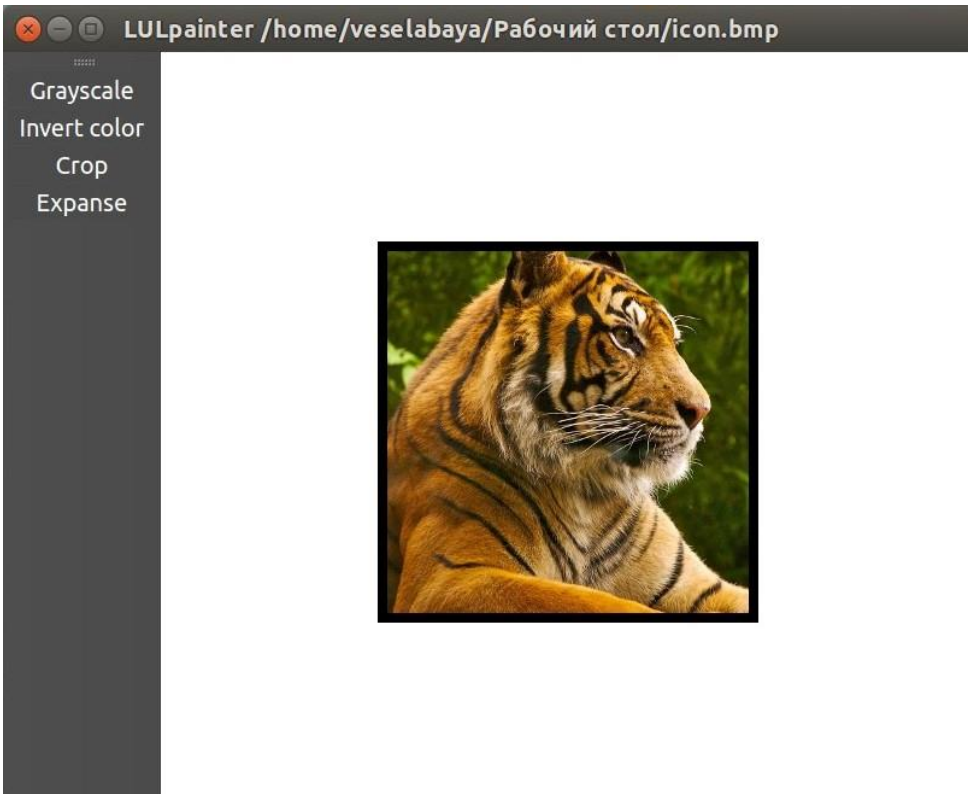
4. Изменение размера изображения путем расширения фона с выбранным цветом заливки.

Графический интерфейс был написан с использованием фреймворка Qt.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

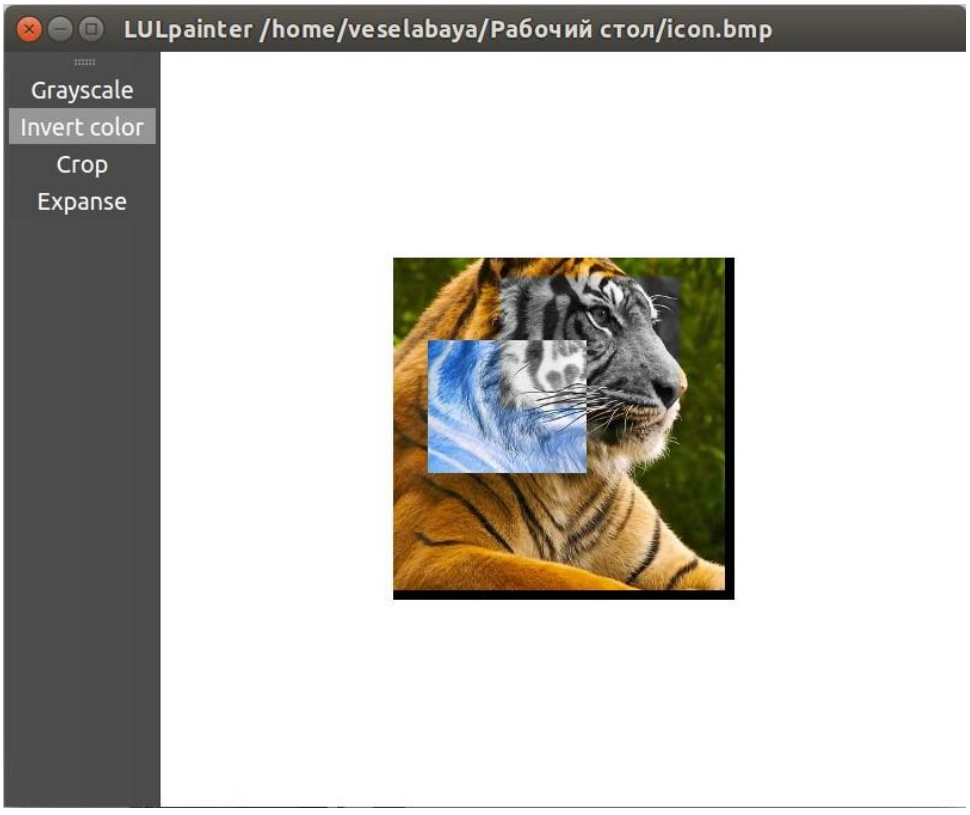
1. Официальный сайт Qt: https://www.qt.io
2. Откуда скачать Qt Creator: https://www.qt.io/qt-features-libraries-apis-tools-and-ide/
3. “Первые шаги” для начинающий на Qt wiki: https://wiki.qt.io/Qt_for_Beginners
4. Видеолекции Кирилла Владимировича по Qt: https://www.youtube.com/playlist?list=PL754BBE9A89BA9D3B
5. Статья о формате BMP: https://ru.wikipedia.org/wiki/BMP

ПРИЛОЖЕНИЕ А.
ПРИМЕРЫ РАБОТЫ

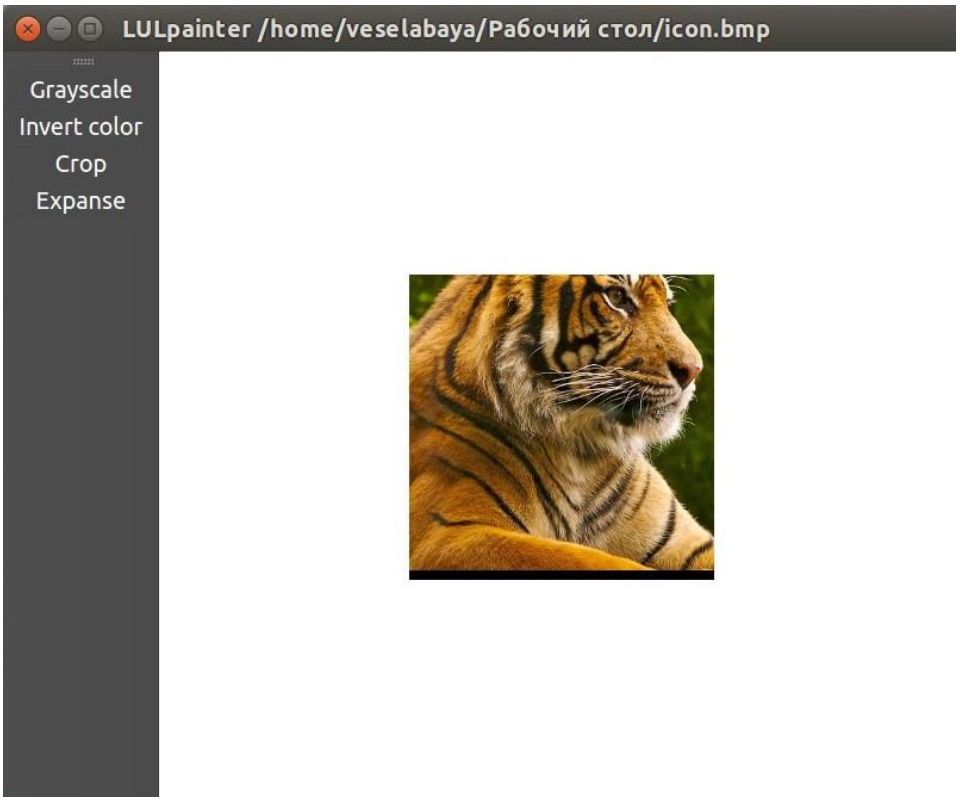


1. Обрезание

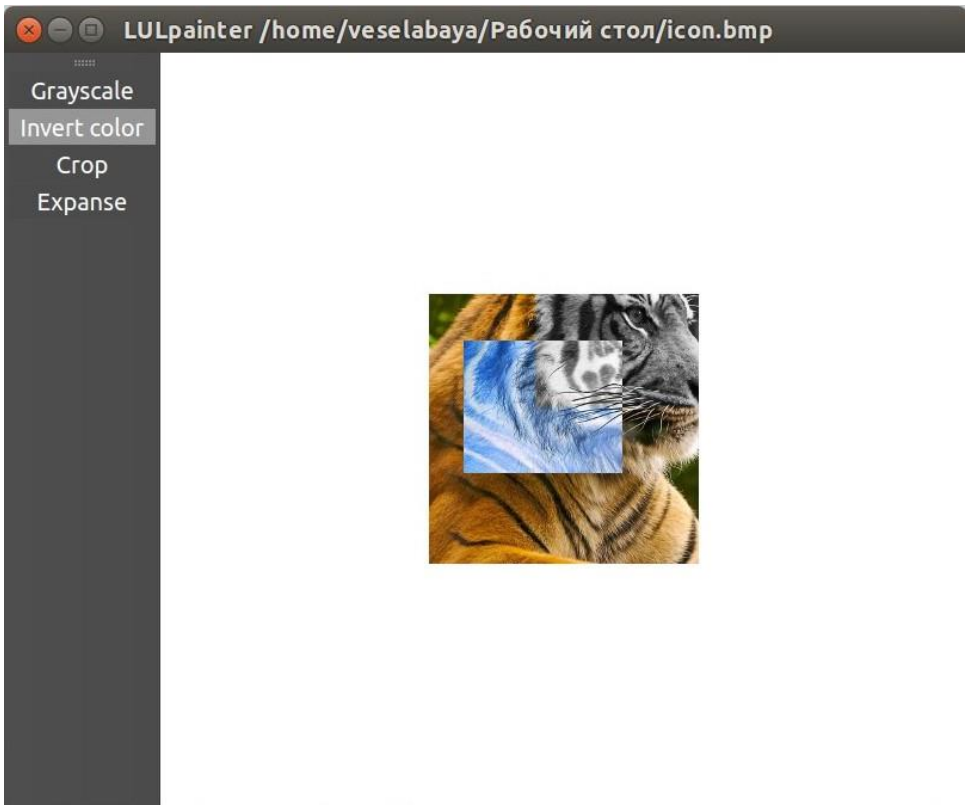
1.1. Левый верхний угол



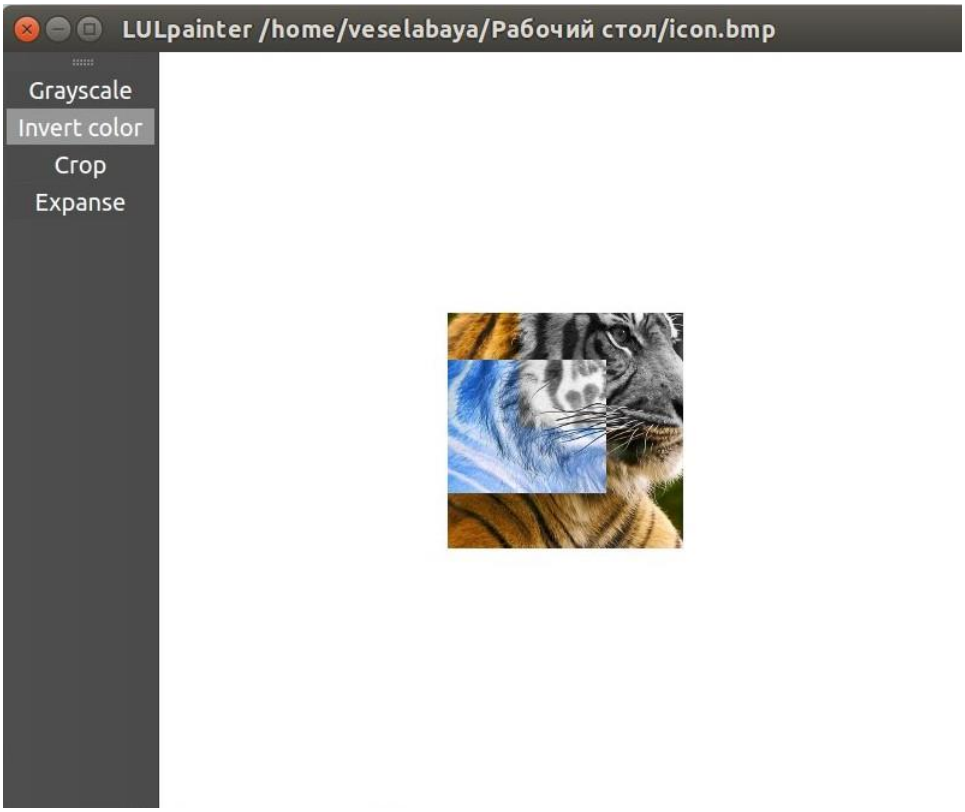
1.2. Правый верхний



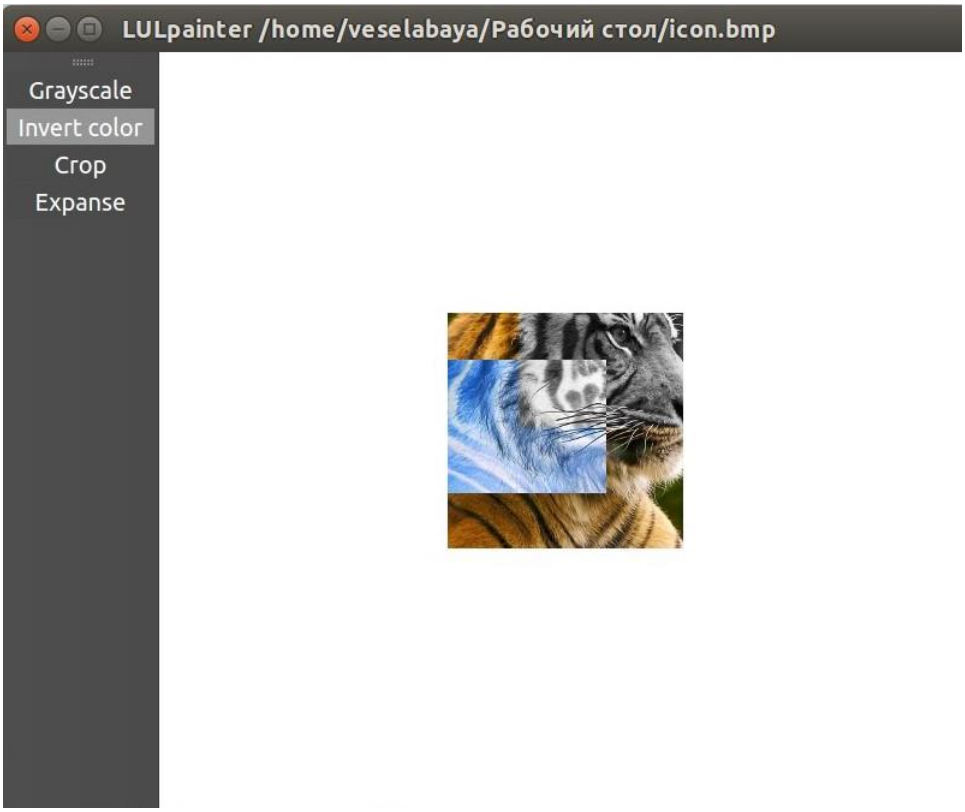
1.3. Правый нижний



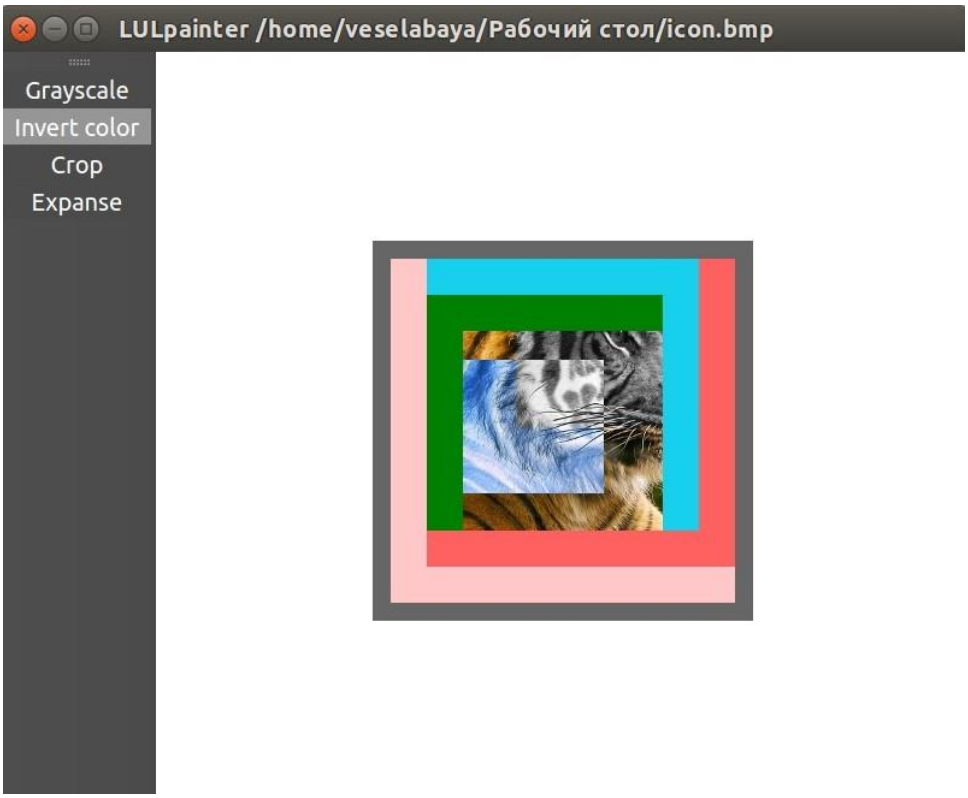
1.4. Левый нижний



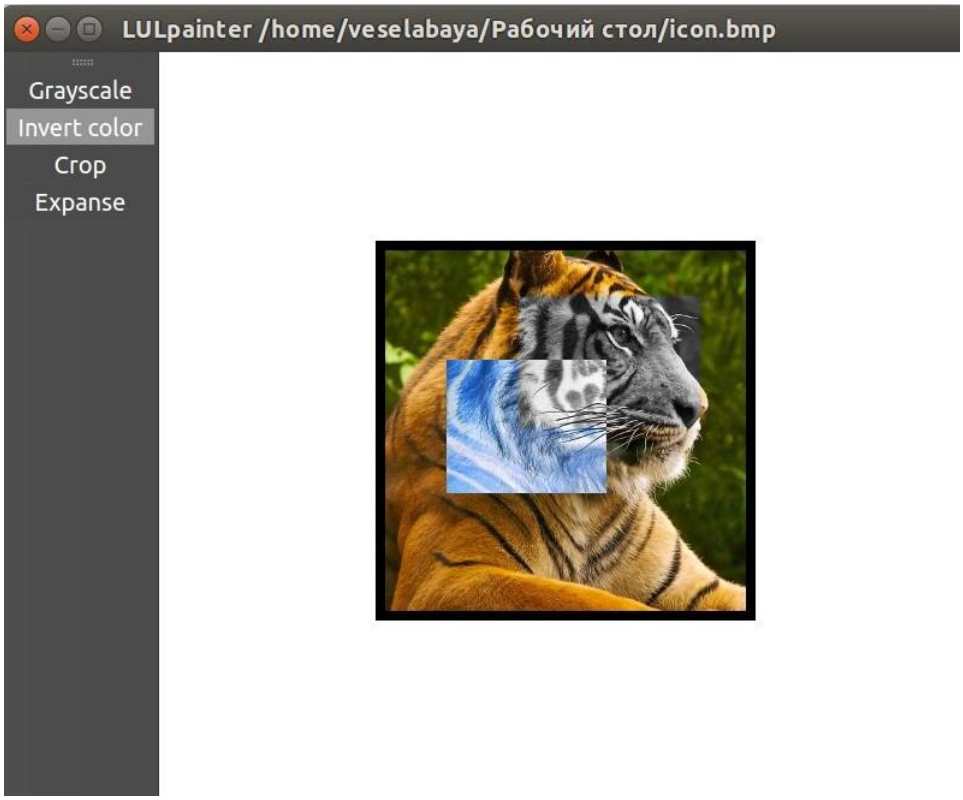
1.5. Центр



2. Расширение



3. Негатив и Grayscale



ПРИЛОЖЕНИЕ Б. ИСХОДНЫЙ КОД

1. Bmp.h

```
#ifndef BMP_H
#define BMP_H

#include "bmp_image24.h"

#include <QMessageBox>
#include <QErrorMessage>

#include <string>
#include <memory>

#define BI_SIZE 40

namespace Bmp {

    Bmp_image* bmp(std::string file_path);

    Bmp_image* copy(Bmp_image* image);

    Bmp_image* create(int width, int height);

    void save(Bmp_image* image, std::string file_path="");
}

#endif // BMP_H
```

2. bmp_exceptions.h

```
#ifndef BMP_EXCEPTIONS_H
#define BMP_EXCEPTIONS_H

#include <exception>

namespace Bmp {
    class Bad_bitcount: public std::exception {
    public:
        Bad_bitcount(int bitcount);
        const char* what() const throw();

        int err_bitcount() const;

    private:
        int bitcount;
    };

    class Bad_size: public std::exception {
    public:
        Bad_size(int width, int height);
        const char* what() const throw();

        int err_width() const;

        int err_height() const;

    private:
```

```
        int width;  /*!< Creating image's width */
        int height; /*!< Creating image's height */
    };

    class Bad_resize: public std::exception {
    public:
        Bad_resize(int vertical, int horizontal);
        const char* what() const throw();

        int err_vertical() const;

        int err_horizontal() const;

    private:
        int vertical;   /*!< vertical resize */
        int horizontal; /*!< horizontal resize */
    };
}

#endif // BMP_EXCEPTIONS_H
```

3. bmp_image.h

```
#ifndef BMP_IMAGE_H
#define BMP_IMAGE_H

#include "resize_direction.h"
#include "bmp_exceptions.h"

#include <QColor>
#include <QImage>

#define BYTE_SIZE 8

#define BM_BITCOUNT_INDEX 28

#pragma pack(push, 1)
struct BitMapFileHeader {
    char           bfType1;     /*!< char for 'B' */
    char           bfType2;     /*!< char for 'M' */
    unsigned       bfSize;      /*!< size of a file */
    unsigned short bfReserved1; /*!< Reserved 2 bytes */
    unsigned short bfReserved2; /*!< Reserved 2 bytes */
    unsigned       bfOffBits;   /*!< Order of byte where raster begins */
};

struct BitMapInfo {
    unsigned       biSize;          /*!< This structure size */
    int            biWidth;         /*!< Image width in px */
    int            biHeight;        /*!< Image height in px */
    unsigned short biPlanes;        /*!< Always value: 1 */
    unsigned short biBitCount;      /*!< Image bitcount */
    unsigned       biCompression;   /*!< Compression flag */
    unsigned       biSizeImage;     /*!< Image size on bytes */
    int            biXPelsPerMeter; /*!< Pixel per meter on x-axis */
    int            biYPelsPerMeter; /*!< Pixel per meter on y-axis */
    unsigned       biClrUsed;       /*!< Palette size in cells */
    unsigned       biClrImportant;  /*!< Amount of cells form begin of palette
to the last cell(including) */
};
#pragma pack(pop)
```

```
class Bmp_image {
public:
    virtual ~Bmp_image() {}

    virtual void set_color(int x, int y, QColor const& color) = 0;

    virtual int get_size() const       = 0;

    virtual int get_height() const     = 0;

    virtual int get_width() const      = 0;

    virtual short get_bitcount() const = 0;

    virtual uint8_t* get_raster() const = 0;

    virtual uint8_t get_raster(int x, int y) const = 0;

    virtual QColor get_color(int x, int y) const = 0;

    virtual QImage get_qImage() const              = 0;

    virtual void invert_color() {}

    virtual void grayscale() {}

    virtual void invert_color(int x1, int y1, int x2, int y2) {}

    virtual void grayscale(int x1, int y1, int x2, int y2) {}

    virtual void crop(int vertical_crop, int horizontal_crop,
Bmp::Resize_direction direction) {}

    virtual void expanse(int vertical_exp, int horizontal_crop,
Bmp::Resize_direction direction, QColor color) {}
};

#endif // BMP_IMAGE_H
```

4. bmp_image24.h

```
 #ifndef BMP_IMAGE24_H
#define BMP_IMAGE24_H

#include "bmp_image.h"

#define ALIGNMENT24(width) (4*(((width)*3)%4 ? 1 : 0) - ((width)*3)%4)

#include <QErrorMessage>

#include <cstdint>   // for uint8_t
#include <algorithm> // for max, min
#include <fstream>
#include <string>

class Bmp_image24: public Bmp_image {
public:
    /*! Constructor
```

```
    Bmp_image24(std::string file_path);

    Bmp_image24(int width, int height, uint8_t* raster=nullptr);

    Bmp_image24(Bmp_image24 const& other);

    Bmp_image24& operator=(Bmp_image24 const& other);

    Bmp_image24(Bmp_image24&& other);

    Bmp_image24& operator=(Bmp_image24&& other);

    ~Bmp_image24();

    // setters
    void set_color(int x, int y, QColor const& color);

    // getters
    int get_size() const;                       // return size in bytes
    int get_width() const;
    int get_height() const;
    short get_bitcount() const;
    uint8_t* get_raster() const;                // return raster like byte array
    uint8_t get_raster(int i, int j) const;     // return raster byte on i-row
and j-column

    QColor get_color(int x, int y) const;       // return color from pixel on
position (row, column)
    QImage get_qImage() const;                  // return QImage from raster
image
    void invert_color();                        // invert whole image
    void grayscale();                           // convert to grayscale image

    virtual void invert_color(int x1, int y1, int x2, int y2); // invert only
rectangle
    virtual void grayscale(int x1, int y1, int x2, int y2);    // grayscale only
rectangle

    virtual void crop(int vertical_crop, int horizontal_crop,
Bmp::Resize_direction direction);
    virtual void expanse(int vertical_exp, int horizontal_crop,
Bmp::Resize_direction direction, QColor color);
private:

    int size;         /*!< Image size in bytes */
    int width;        /*!< Image width in px */
    int height;       /*!< Image height in px */
    short bitcount;   /*!< Image bitcount in bits/color */
    uint8_t** raster; /*!< Raster image */
};

#endif // BMP_IMAGE24_H
```

5. crop_dialog.h

```
#ifndef CROP_DIALOG_H
#define CROP_DIALOG_H

#include "resize_direction.h"
```

```
#include <QDialog>
#include <QSettings>

namespace Ui {
    class Crop_dialog;
}

class Crop_dialog: public QDialog {
    Q_OBJECT

public:
    explicit Crop_dialog(int max_width, int max_height, QWidget *parent = 0);
    ~Crop_dialog();

signals:
    void ok_button_clicked(int vertical_crop, int horizontal_crop,
Bmp::Resize_direction direction);

private slots:
    void write_settings();
    void read_settings();

    void on_cancel_button_clicked();
    void on_ok_button_clicked();

private:
    Ui::Crop_dialog *ui;
};

#endif // CROP_DIALOG_H
```

6. expanse_dialog.h

#ifndef EXPANSE_DIALOG_H

```
#define EXPANSE_DIALOG_H

#include "resize_direction.h"

#include <QDialog>
#include <QColorDialog>
#include <QSettings>
#include <QColor>

namespace Ui {
    class Expanse_dialog;
}

class Expanse_dialog: public QDialog {
    Q_OBJECT

public:
    explicit Expanse_dialog(QWidget *parent = 0);
    ~Expanse_dialog();

signals:
    void ok_button_clicked(int vertical_crop, int horizontal_crop,
Bmp::Resize_direction direction, QColor color);

private slots:
    void write_settings();
```

```
    void read_settings();
    void set_background_color(QColor color);

    void on_cancel_button_clicked();
    void on_ok_button_clicked();

    void on_color_button_clicked();

private:
    Ui::Expanse_dialog *ui;
    QColor color; /*!< Fill color */
};

#endif // EXPANSE_DIALOG_H
```

7. help_dialog.h

#ifndef HELP_DIALOG_H

```
#define HELP_DIALOG_H

#include <QDialog>

namespace Ui {
    class Help_dialog;
}

class Help_dialog : public QDialog {
    Q_OBJECT

public:
    explicit Help_dialog(QWidget *parent = 0);
    ~Help_dialog();

private:
    Ui::Help_dialog *ui;
};

#endif // HELP_DIALOG_H
```

8. info_dialog.h

#ifndef INFO_DIALOG_H

```
#define INFO_DIALOG_H

#include <bmp.h>

#include <QDialog>
#include <QLabel>
#include <QVBoxLayout>
#include <QHBoxLayout>

class Info_dialog: public QDialog {
    Q_OBJECT

public:
    explicit Info_dialog(Bmp_image const* image);
    ~Info_dialog();

public slots:
    void change_info(Bmp_image const* image);
```

```
    static QString make_beauty(int size);

private:
    QLabel* width;     /*!< Image width */
    QLabel* height;    /*!< Image height */
    QLabel* size;      /*!< Image size */
    QLabel* bitcount; /*!< Image bitcount */
};

#endif // INFO_DIALOG_H
```

9. mainwindow.h

#ifndef MAINWINDOW_H

```
#define MAINWINDOW_H

#include "bmp.h"
#include "save_dialog.h"
#include "my_graphics_scene.h"
#include "crop_dialog.h"
#include "expanse_dialog.h"
#include "help_dialog.h"
#include "info_dialog.h"

#include <QMainWindow>
#include <QSettings>
#include <QFileDialog>
#include <QGraphicsScene>
#include <QPixmap>
#include <QImage>
#include <QString>
#include <QCloseEvent>
#include <QHBoxLayout>
#include <QVBoxLayout>
#include <QInputDialog>

#include <algorithm>

namespace Ui { class MainWindow; }

class MainWindow: public QMainWindow {
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

signals:
    void image_changed(Bmp_image const* image);

private slots:
    void read_settings();  // read config (size and position of window, path to
last opened image)
    void write_settings(); // write this config
    void closeEvent(QCloseEvent* event);
    bool cancel_toggle();
    void grayscale_toggle();
    void invert_toggle();
    void mouseReleased();
    void crop_image(int vertical_crop, int horizontal_crop,
Bmp::Resize_direction crop_direction);
```

```
    void expanse_image(int vertical_exp, int horizontal_exp,
Bmp::Resize_direction expanse_direction, QColor color);
    void save_as(QWidget* parent=nullptr);

    void on_actionOpen_triggered();
    void on_actionSave_triggered();
    void on_actionSave_As_triggered();
    void on_actioncoordinates_gray_triggered();
    void on_actioncoordinates_invert_triggered();
    void on_actionGraysacle_triggered();
    void on_actionInvert_triggered();
    void on_actionQuit_triggered();
    void on_actionNew_triggered();
    void on_actionCrop_3_triggered();
    void on_actionExpanse_triggered();
    void on_actionImage_info_triggered();
    void on_actionHelp_triggered();

private:
    Ui::MainWindow* ui;
    My_graphics_scene* scene; /*!< Scene with opened image */
    Bmp_image* bmp_image;     /*!< Pointer on image */

    QString open_file_path;   /*!< Opened image file path */
    QString prev_file_path;   /*!< Pevious image file path */
    // flags
    bool grayscale_clicked;   /*!< Grayscale tool choosen */
    bool invert_clicked;      /*!< Invert colot tool choosen */
    bool cancel_clicked;      /*!< Cancel button in save dialog clicked */
    bool changed;             /*!< Image has been changed */
};

#endif // MAINWINDOW_H
```

10. my_graphics_scene.h

```
#ifndef MY_GRAPHICS_SCENE_H
#define MY_GRAPHICS_SCENE_H

#include <QGraphicsScene>
#include <QGraphicsSceneMouseEvent>
#include <QPoint>

class My_graphics_scene: public QGraphicsScene {
    Q_OBJECT

public:
    explicit My_graphics_scene(QObject* parent=nullptr);
    ~My_graphics_scene();

signals:
    void mouseMoved();
    void mousePressed();
    void mouseReleased();

public slots:
    QPoint get_first();

    QPoint get_last();

private slots:
```

```
    void mouseMoveEvent(QGraphicsSceneMouseEvent* event) override;
    void mousePressEvent(QGraphicsSceneMouseEvent* event) override;
    void mouseReleaseEvent(QGraphicsSceneMouseEvent* event) override;

private:
    QPoint first; /*!< start point */
    QPoint last;  /*!< finish point */
};

#endif // MY_GRAPHICS_SCENE_H
```

11. resize_direction.h

```
#ifndef RESIZE_DIRECTION_H
#define RESIZE_DIRECTION_H

namespace Bmp {
    enum class Resize_direction {center,
                                 upper_left,
                                 upper_right,
                                 lower_right,
                                 lower_left
                                };
}

#endif // RESIZE_DIRECTION_H
```

12. save_dialog.h

```
#ifndef SAVE_DIALOG_H
#define SAVE_DIALOG_H

#include <QDialog>
#include <QCloseEvent>
#include <QLabel>
#include <QString>

namespace Ui {
    class save_quit_dialog;
}

class Save_dialog: public QDialog {
    Q_OBJECT

public:
    explicit Save_dialog(QWidget *parent = 0);
    ~Save_dialog();

signals:
    void save_button_clicked(QWidget* save_dialog);

    void cancel_button_clicked(bool clicked);

private slots:
    void closeEvent(QCloseEvent* event);
    void on_save_button_clicked();
    void on_cancel_button_clicked();
    void on_no_button_clicked();

private:
```

```
    Ui::save_quit_dialog *ui;
};

#endif // SAVE_DIALOG_H
```

13. bmp.cpp

```
#include "bmp.h"
Bmp_image* Bmp::bmp(std::string file_path) {
    if (file_path != "") {
        std::ifstream file(file_path, std::ios::in | std::ios::binary);
        if (file.is_open()) {
            char check[3]{};
            file.read(check, 2);

            if (std::string(check) == "BM") {
                // bitcount reading
                short bitcount;
                file.seekg(BM_BITCOUNT_INDEX);
                file.read(reinterpret_cast<char*>(&bitcount), sizeof(short));
                file.close();

                Bmp_image* bmp_image = nullptr;
                switch(static_cast<int>(bitcount)) {
                    case 24:
                        try {
                            bmp_image = new Bmp_image24(file_path);
                        } catch (std::bad_alloc& e) {
                            QErrorMessage err_msg;
                            err_msg.setWindowTitle("Bad allocation");
                            err_msg.showMessage(QString("Not enough memory to do
this operation"));
                            err_msg.exec();

                            bmp_image = nullptr;
                        } break;
                    default:
                        throw Bmp::Bad_bitcount{static_cast<int>(bitcount)};
                }

                return bmp_image;
            }
        }
    }

    return nullptr;
}

Bmp_image* Bmp::copy(Bmp_image *image) {
    Bmp_image* copy;
    switch(image->get_bitcount()) {
        case 24: copy = new Bmp_image24(*static_cast<Bmp_image24*>(image));
break;

    }

    return copy;
}

Bmp_image* Bmp::create(int width, int height) {
    Bmp_image* image = new Bmp_image24(width, height);
```

```
    return image;
}

void Bmp::save(Bmp_image *image, std::string file_path) {
    std::ofstream file(file_path, std::ios::out | std::ios::binary);

    if (file.is_open()) {
        BitMapFileHeader bm_header;
        BitMapInfo       bm_info;

        bm_header.bfType1     = 'B';
        bm_header.bfType2     = 'M';
        bm_header.bfSize      = sizeof(BitMapFileHeader) +
                                sizeof(BitMapInfo) +
                                image->get_size();
        bm_header.bfReserved1 = bm_header.bfReserved2 = 0;

        bm_info.biSize          = BI_SIZE;
        bm_info.biWidth         = image->get_width();
        bm_info.biHeight        = image->get_height();
        bm_info.biPlanes        = 1;
        bm_info.biCompression   = 0;
        bm_info.biSizeImage     = image->get_size();
        bm_info.biXPelsPerMeter = 0;
        bm_info.biYPelsPerMeter = 0;

        switch (image->get_bitcount()) {
            case 24:
                bm_header.bfOffBits    = 54;
                bm_info.biBitCount     = 24;
                bm_info.biClrUsed      = 0;
                bm_info.biClrImportant = 0;

                file.write(reinterpret_cast<char*>(&bm_header),
sizeof(BitMapFileHeader));
                file.write(reinterpret_cast<char*>(&bm_info),
sizeof(BitMapInfo));

                uint8_t** saving_raster = new uint8_t*[bm_info.biHeight];
                saving_raster[0] = new uint8_t[bm_info.biSizeImage]{};
                for (int i = 1; i != bm_info.biHeight; ++i) {
                    saving_raster[i] = saving_raster[i-1] + (bm_info.biWidth*3)
+ ALIGNMENT24(bm_info.biWidth);
                }

                for (int i = 0; i != bm_info.biHeight; ++i) {
                    for (int j = 0; j != bm_info.biWidth*3; ++j) {
                        saving_raster[i][j] = image->get_raster(i, j);
                    }
                }

                file.write(reinterpret_cast<char*>(saving_raster[0]),
                                                    bm_info.biHeight *
((bm_info.biWidth*3) + ALIGNMENT24(bm_info.biWidth)));
                file.close();

                delete [] saving_raster[0];
                delete [] saving_raster;
            break;
        }
    } else {
```

```
        QErrorMessage err_msg;
        err_msg.setWindowTitle("Saving file error");
        err_msg.showMessage("Can't save this file on some reason.");
        err_msg.setStyleSheet("QPushButton {"
                              "    color: white;"
                              "    background-color: rgb(75, 75, 75);"
                              "}");
        err_msg.exec();
    }
}
```

14. bmp_exceptions.cpp

```
#include "bmp_exceptions.h"
Bmp::Bad_bitcount::Bad_bitcount(int bitcount): std::exception{},
                                               bitcount(bitcount) {}

const char* Bmp::Bad_bitcount::what() const throw() {
    return "This version supports only 24 bitcount";
}

int Bmp::Bad_bitcount::err_bitcount() const {
    return bitcount;
}

Bmp::Bad_size::Bad_size(int width, int height): std::exception{},
                                                width(width), height(height) {}

const char* Bmp::Bad_size::what() const throw() {
    return "Incorrect image's raster size";
}

int Bmp::Bad_size::err_width() const {
    return width;
}

int Bmp::Bad_size::err_height() const {
    return height;
}

Bmp::Bad_resize::Bad_resize(int vertical, int horizontal): std::exception{},
                                                           vertical(vertical),
horizontal(horizontal) {}

const char* Bmp::Bad_resize::what() const throw() {
    return "Incorrect resize parameters";
}

int Bmp::Bad_resize::err_vertical() const {
    return vertical;
}

int Bmp::Bad_resize::err_horizontal() const {
    return horizontal;
}
```

15. bmp_image24.cpp

```
#include "bmp_image24.h"
Bmp_image24::Bmp_image24(std::string file_path) {
    if (file_path != "") {
        BitMapFileHeader bm_header;
        BitMapInfo       bm_info;

        std::ifstream file(file_path, std::ios::in | std::ios::binary);
        if (file.is_open()) {
            file.read(reinterpret_cast<char*>(&bm_header),
sizeof(BitMapFileHeader));
            file.read(reinterpret_cast<char*>(&bm_info), sizeof(BitMapInfo));

            width    = bm_info.biWidth;
            height   = bm_info.biHeight;
            size     = bm_info.biSizeImage;
            bitcount = bm_info.biBitCount;

            // allocation memory for raster image
            raster = new uint8_t*[height];
            raster[0] = new uint8_t[size];
            for (int i = 1; i != height; ++i) {
                raster[i] = raster[i-1] + ((width*3) + ALIGNMENT24(width));
            }

            file.seekg(bm_header.bfOffBits); // here raster starts
            for (int i = 0; i != height; ++i) {
                for (int j = 0; j != width*3 + ALIGNMENT24(width); ++j) {
                    file.read(reinterpret_cast<char*>(&raster[i][j]), 1);
                }
            }

            file.close();
        }
    }
}

Bmp_image24::Bmp_image24(int width, int height, uint8_t* raster): size(height *
((width*3) + ALIGNMENT24(width))), // 24 - bitcount
                                                                  width(width),

height(height),
                                                                  bitcount(24) {
    if (width < 0 || height < 0) {
        throw Bmp::Bad_size(width, height);
    }

    this->raster = new uint8_t*[height];
    this->raster[0] = new uint8_t[size]{};
    for (int i = 1; i != height; ++i) {
        this->raster[i] = this->raster[i-1] + ((width*3) + ALIGNMENT24(width));
    }

    if (raster) {
        for (int i = 0; i != height; ++i) {
            for (int j = 0; j != (width*3); ++j) {
                this->raster[i][j] = raster[i*(width*3) + j];
            }
        }
    } else {
        for (int i = 0; i != height; ++i) {
```

```
            // null init for all bytes in new raster
            for (int j = 0; j != ((width*3) + ALIGNMENT24(width)); ++j) {
                this->raster[i][j] = 255;
            }
        }
    }
}

Bmp_image24::Bmp_image24(Bmp_image24 const& other) {
    size     = other.size;
    width    = other.width;
    height   = other.height;
    bitcount = other.bitcount;

    // allocation
    raster = new uint8_t*[height];
    raster[0] = new uint8_t[size];
    for (int i = 1; i != height; ++i) {
        raster[i] = raster[i-1] + ((width*3) + ALIGNMENT24(width));
    }

    for (int i = 0; i != height; ++i) {
        for (int j = 0; j != (width*3) + ALIGNMENT24(width); ++j) {
            raster[i][j] = other.raster[i][j];
        }
    }
}

Bmp_image24& Bmp_image24::operator=(Bmp_image24 const& other) {
    delete [] raster;
    delete [] raster[0];

    size     = other.size;
    width    = other.width;
    height   = other.height;
    bitcount = other.bitcount;

    // allocation
    raster = new uint8_t*[other.get_height()];
    raster[0] = new uint8_t[other.get_size()];
    for (int i = 0; i != height; ++i) {
        for (int j = 0; j != (width*3) + ALIGNMENT24(width); ++j) {
            raster[i][j] = other.raster[i][j];
        }
    }

    return *this;
}

//-////////////////////////////////////////////////////////////
// displacement

Bmp_image24::Bmp_image24(Bmp_image24&& other): size(other.size),
                                               width(other.width),
                                               height(other.height),
                                               bitcount(other.bitcount) {
    other.size     = 0;
    other.width    = 0;
    other.height   = 0;
    other.bitcount = 0;
```

```
    // allocation
    raster = new uint8_t*[other.get_height()];
    raster[0] = new uint8_t[other.get_size()];
    for (int i = 0; i != height; ++i) {
        for (int j = 0; j != (width*3) + ALIGNMENT24(width); ++j) {
            raster[i][j] = other.raster[i][j];
        }
    }

    other.raster = nullptr;
}

Bmp_image24& Bmp_image24::operator=(Bmp_image24&& other) {
    delete [] raster[0];
    delete [] raster;

    size     = other.size;
    height   = other.height;
    width    = other.width;
    bitcount = other.bitcount;

    other.size     = 0;
    other.width    = 0;
    other.height   = 0;
    other.bitcount = 0;

    // allocation
    raster = new uint8_t*[other.get_height()];
    raster[0] = new uint8_t[other.get_size()];
    for (int i = 0; i != height; ++i) {
        for (int j = 0; j != (width*3) + ALIGNMENT24(width); ++j) {
            raster[i][j] = other.raster[i][j];
        }
    }

    other.raster = nullptr;
    return *this;
}

Bmp_image24::~Bmp_image24() {
    delete [] raster[0];
    delete [] raster;
}

//-////////////////////////////////////////////////////////////

void Bmp_image24::set_color(int x, int y, QColor const& color) {
    raster[height-y-1][x*3]   = color.blue(); // this magic because of invert
order in bmp raster
    raster[height-y-1][x*3+1] = color.green();
    raster[height-y-1][x*3+2] = color.red();
}

//-////////////////////////////////////////////////////////////

int Bmp_image24::get_size() const {
    return size;
}

int Bmp_image24::get_height() const {
```

```
    return height;
}

int Bmp_image24::get_width() const {
    return width;
}

short Bmp_image24::get_bitcount() const {
    return bitcount;
}

uint8_t* Bmp_image24::get_raster() const {
    uint8_t* copy;
    copy = new uint8_t[height * width*3];
    for (int i = 0; i != height; ++i) {
        for (int j = 0; j != width*3; ++j) {
            copy[i*(width*3) + j] = raster[i][j];
        }
    }

    return copy;
}

uint8_t Bmp_image24::get_raster(int i, int j) const {
    return raster[i][j];
}

QColor Bmp_image24::get_color(int x, int y) const {
    return qRgb(raster[height-y-1][x*3 + 2], raster[height-y-1][x*3 + 1],
raster[height-y-1][x*3]); // this magic because of invert order in bmp raster
}

QImage Bmp_image24::get_qImage() const {
    QImage image(width, height, QImage::Format_RGB888); // this format for 24-
bitcount (8,8,8)
    for (int i = 0; i != height; ++i) {
        for (int j = width-1; j >= 0; --j) {
            image.setPixelColor(j, i, this->get_color(j, i));
        }
    }

    return image;
}

//-////////////////////////////////////////////////////////////

void Bmp_image24::invert_color() {
    for (int i = 0; i != height; ++i) {
        for (int j = 0; j != (width*3) + ALIGNMENT24(width); ++j) {
            raster[i][j] = 255 - raster[i][j];
        }
    }
}

void Bmp_image24::grayscale() {
    for (int i = 0; i != height; ++i) {
        for (int j = 0; j < (width*3) + ALIGNMENT24(width) - 3; j+=3) {
            uint8_t luma = 0.2126*raster[i][j] + 0.7152*raster[i][j+1] +
0.0722*raster[i][j+2]; // https://habrahabr.ru/post/304210/
            raster[i][j] = luma;
            raster[i][j+1] = luma;
```

```
            raster[i][j+2] = luma;
        }
    }
}

void Bmp_image24::invert_color(int x1, int y1, int x2, int y2) {
    // find upper left and lower right corners coordinates
    // in case of out of image
    if (x1 < 0) x1 = 0;
    if (y1 < 0) y1 = 0;
    if (x2 < 0) x2 = 0;
    if (y2 < 0) y2 = 0;

    int x_min = std::min(std::min(x1, x2), width);
    int y_min = std::min(std::min(y1, y2), height);
    int x_max = std::min(std::max(x1, x2), width);  // in case of std::max() out
of image
    int y_max = std::min(std::max(y1, y2), height);

    for (int y = y_min; y != y_max; ++y) {
        for (int x = x_min; x != x_max; ++x) {
            this->set_color(x, y, QColor(255 - raster[height-y-1][x*3+2], //
this magic because of invert order in bmp raster
                                         255 - raster[height-y-1][x*3+1],
                                         255 - raster[height-y-1][x*3]));
        }
    }
}

void Bmp_image24::grayscale(int x1, int y1, int x2, int y2) {
    // find upper left and lower right corners coordinates
    // in case of out of image
    if (x1 < 0) x1 = 0;
    if (y1 < 0) y1 = 0;
    if (x2 < 0) x2 = 0;
    if (y2 < 0) y2 = 0;

    int x_min = std::min(std::min(x1, x2), width);
    int y_min = std::min(std::min(y1, y2), height);
    int x_max = std::min(std::max(x1, x2), width);
    int y_max = std::min(std::max(y1, y2), height);

    for (int y = y_min; y != y_max; ++y) {
        for (int x = x_min; x != x_max; ++x) {
            QColor rgb = get_color(x, y);
            uint8_t luma = 0.2126*rgb.red() + 0.7152*rgb.green() +
0.0722*rgb.blue(); // https://habrahabr.ru/post/304210/
            set_color(x, y, QColor(luma, luma, luma));
        }
    }
}

void Bmp_image24::crop(int vertical_crop, int horizontal_crop,
Bmp::Resize_direction direction) {
    if (vertical_crop < 0 || horizontal_crop < 0) {
        throw Bmp::Bad_resize{vertical_crop, horizontal_crop};
    }

    height -= vertical_crop;
    width -= horizontal_crop;
    size = height * (width*3 + ALIGNMENT24(width));
```

```
    uint8_t** cropped_raster = new uint8_t*[height];
    cropped_raster[0] = new uint8_t[size]{};
    for (int i = 1; i != height; ++i) {
        cropped_raster[i] = cropped_raster[i-1] + ((width*3) +
ALIGNMENT24(width));
    }

    switch (direction) {
        case Bmp::Resize_direction::center:
            vertical_crop /= 2;
            horizontal_crop /= 2;
            for (int i = 0; i != height; ++i) {
                for (int j = 0; j != width*3 + ALIGNMENT24(width); ++j) {
                    cropped_raster[i][j] =
raster[i+vertical_crop][j+horizontal_crop * 3];
                }
            } break;

        case Bmp::Resize_direction::upper_left:
            for (int i = 0; i != height; ++i) {
                for (int j = 0; j != width*3 + ALIGNMENT24(width); ++j) {
                    cropped_raster[i][j] = raster[i][j+horizontal_crop*3];
                }
            } break;

        case Bmp::Resize_direction::upper_right:
            for (int i = 0; i != height; ++i) {
                for (int j = 0; j != width*3 + ALIGNMENT24(width); ++j) {
                    cropped_raster[i][j] = raster[i][j];
                }
            } break;

        case Bmp::Resize_direction::lower_right:
            for (int i = 0; i != height; ++i) {
                for (int j = 0; j != width*3 + ALIGNMENT24(width); ++j) {
                    cropped_raster[i][j] = raster[i+vertical_crop][j];
                }
            } break;

        case Bmp::Resize_direction::lower_left:
            for (int i = 0; i != height; ++i) {
                for (int j = 0; j != width*3 + ALIGNMENT24(width); ++j) {
                    cropped_raster[i][j] =
raster[i+vertical_crop][j+horizontal_crop*3];
                }
            } break;
    }

    delete [] raster[0];
    delete [] raster;

    raster = cropped_raster;
    raster[0] = cropped_raster[0];
}

void Bmp_image24::expanse(int vertical_exp, int horizontal_exp,
Bmp::Resize_direction direction, QColor color) {
    if (vertical_exp < 0 || horizontal_exp < 0) {
        throw Bmp::Bad_resize{vertical_exp, horizontal_exp};
    }
```

```
    int old_width = width;
    int old_height = height;

    height += vertical_exp;
    width += horizontal_exp;
    size = height * (width*3 + ALIGNMENT24(width));

    uint8_t** expansed_raster = new uint8_t*[height];
    expansed_raster[0] = new uint8_t[size]{};
    for (int i = 1; i != height; ++i) {
        expansed_raster[i] = expansed_raster[i-1] + ((width*3) +
ALIGNMENT24(width));
    }

    switch(direction) {
        case Bmp::Resize_direction::center:
            vertical_exp /= 2;
            horizontal_exp /= 2;
            for (int i = 0; i != vertical_exp; ++i) {
               for (int j = 0; j < (width*3) - 2; j += 3) {
                   expansed_raster[i][j] = color.blue();
                   expansed_raster[i][j+1] = color.green();
                   expansed_raster[i][j+2] = color.red();
               }
            }

            for (int i = vertical_exp; i != old_height + vertical_exp; ++i) {
                for (int j = 0; j < horizontal_exp*3 - 2; j += 3) {
                    expansed_raster[i][j] = color.blue();
                    expansed_raster[i][j+1] = color.green();
                    expansed_raster[i][j+2] = color.red();
                }

                for (int j = horizontal_exp*3; j != old_width*3 +
horizontal_exp*3; ++j) {
                    expansed_raster[i][j] = raster[i-vertical_exp][j-
horizontal_exp*3];
                }

                for (int j = old_width*3 + horizontal_exp*3; j < width*3 - 2; j
+= 3) {
                    expansed_raster[i][j] = color.blue();
                    expansed_raster[i][j+1] = color.green();
                    expansed_raster[i][j+2] = color.red();
                }
            }

            for (int i = old_height + vertical_exp; i != height; ++i) {
                for (int j = 0; j < (width*3) - 2; j += 3) {
                    expansed_raster[i][j] = color.blue();
                    expansed_raster[i][j+1] = color.green();
                    expansed_raster[i][j+2] = color.red();
                }
            } break;

        case Bmp::Resize_direction::upper_left:
            for (int i = 0; i != old_height; ++i) {
                for (int j = 0; j < horizontal_exp*3 - 2; j += 3) {
                    expansed_raster[i][j] = color.blue();
                    expansed_raster[i][j+1] = color.green();
                    expansed_raster[i][j+2] = color.red();
```

```
                }

                for (int j = horizontal_exp*3; j != (width*3) +
ALIGNMENT24(width); ++j) {
                    expansed_raster[i][j] = raster[i][j-horizontal_exp*3]; //
TODO some COLOR instead 0
                }
            }
            for (int i = old_height; i != height; ++i) {
                for (int j = 0; j < width*3 - 2; j += 3) {
                    expansed_raster[i][j] = color.blue();
                    expansed_raster[i][j+1] = color.green();
                    expansed_raster[i][j+2] = color.red();
                }
            } break;

        case Bmp::Resize_direction::upper_right:
            for (int i = 0; i != old_height; ++i) {
                for (int j = 0; j != (old_width)*3; ++j) {
                    expansed_raster[i][j] = raster[i][j];
                }

                for (int j = old_width*3; j < width*3 - 2; j += 3) {
                    expansed_raster[i][j] = color.blue();
                    expansed_raster[i][j+1] = color.green();
                    expansed_raster[i][j+2] = color.red();
                }
            }

            for (int i = old_height; i != height; ++i) {
                for (int j = 0; j < width*3 - 2; j += 3) {
                    expansed_raster[i][j] = color.blue();
                    expansed_raster[i][j+1] = color.green();
                    expansed_raster[i][j+2] = color.red();
                }
            } break;

        case Bmp::Resize_direction::lower_right:
            for (int i = 0; i != height - old_height; ++i) {
                for (int j = 0; j < width*3 - 2; j += 3) {
                    expansed_raster[i][j] = color.blue();
                    expansed_raster[i][j+1] = color.green();
                    expansed_raster[i][j+2] = color.red();
                }
            }

            for (int i = height - old_height; i != height; ++i) {
                for (int j = 0; j != old_width*3; ++j) {
                    expansed_raster[i][j] = raster[i-vertical_exp][j];
                }

                for (int j = old_width*3; j < width*3 - 2; j += 3) {
                    expansed_raster[i][j] = color.blue();
                    expansed_raster[i][j+1] = color.green();
                    expansed_raster[i][j+2] = color.red();
                }
            } break;

        case Bmp::Resize_direction::lower_left:
            for (int i = 0; i != height - old_height; ++i) {
                for (int j = 0; j < (width*3) - 2; j += 3) {
```

```
                    expansed_raster[i][j] = color.blue();
                    expansed_raster[i][j+1] = color.green();
                    expansed_raster[i][j+2] = color.red();
                }
            }

            for (int i = height - old_height; i != height; ++i) {
                for (int j = 0; j < horizontal_exp*3 - 2; j += 3) {
                    expansed_raster[i][j] = color.blue();
                    expansed_raster[i][j+1] = color.green();
                    expansed_raster[i][j+2] = color.red();
                }

                for (int j = horizontal_exp*3; j != (width*3) +
ALIGNMENT24(width); ++j) {
                    expansed_raster[i][j] = raster[i-vertical_exp][j -
horizontal_exp*3];
                }
            } break;
    }

    delete [] raster[0];
    delete [] raster;

    raster = expansed_raster;
    raster[0] = expansed_raster[0];
}
```

16. crop_dialog.cpp

```
#include "crop_dialog.h"
#include "ui_crop_dialog.h"

Crop_dialog::Crop_dialog(int max_width, int max_height, QWidget *parent):
QDialog(parent),
                                            ui(new Ui::Crop_dialog) {
    ui->setupUi(this);
    ui->vertical_crop->setMaximum(max_height-1);
    ui->horizontal_crop->setMaximum(max_width-1);
    setWindowTitle("Crop");

    read_settings();
}

Crop_dialog::~Crop_dialog() {
    delete ui;
}

void Crop_dialog::on_cancel_button_clicked() {
    write_settings();
    close();
}

void Crop_dialog::on_ok_button_clicked() {
    Bmp::Resize_direction crop_direction;
    switch(ui->crop_direction->currentIndex()) {
        case 0: crop_direction = Bmp::Resize_direction::center; break;
        case 1: crop_direction = Bmp::Resize_direction::upper_left; break;
        case 2: crop_direction = Bmp::Resize_direction::upper_right; break;
        case 3: crop_direction = Bmp::Resize_direction::lower_right; break;
        case 4: crop_direction = Bmp::Resize_direction::lower_left; break;
```

```
    }

    emit ok_button_clicked(ui->vertical_crop->value(),
                           ui->horizontal_crop->value(),
                           crop_direction);

    write_settings();
    close();
}

void Crop_dialog::write_settings() {
    QSettings settings("My Soft", "LULpainter");

    settings.beginGroup("Crop dialog");
    settings.setValue("vertical_crop", ui->vertical_crop->value());
    settings.setValue("horizontal_crop", ui->horizontal_crop->value());
    settings.setValue("crop_direction", ui->crop_direction->currentIndex());
    settings.setValue("size", size());
    settings.setValue("pos", pos());
    settings.endGroup();
}

void Crop_dialog::read_settings() {
    QSettings settings("My Soft", "LULpainter");

    settings.beginGroup("Crop dialog");
    ui->vertical_crop->setValue(settings.value("vertical_crop", 0).toInt());
    ui->horizontal_crop->setValue(settings.value("horizontal_crop", 0).toInt());
    ui->crop_direction->setCurrentIndex(settings.value("crop_direction",
0).toInt());
    resize(settings.value("size", QSize(403, 211)).toSize());
    move(settings.value("pos", QPoint(200, 200)).toPoint());
    settings.endGroup();
}
```

17. expanse_dialog.cpp

```
 #include "expanse_dialog.h"
#include "ui_expanse_dialog.h"

Expanse_dialog::Expanse_dialog(QWidget *parent): QDialog(parent),
                                                 ui(new Ui::Expanse_dialog) {
    ui->setupUi(this);
    setWindowTitle("Expanse");

    read_settings();
}

Expanse_dialog::~Expanse_dialog() {
    delete ui;
}

void Expanse_dialog::on_cancel_button_clicked() {
    write_settings();
    close();
}

void Expanse_dialog::on_ok_button_clicked() {
    Bmp::Resize_direction expanse_direction;
    switch(ui->crop_direction->currentIndex()) {
        case 0: expanse_direction = Bmp::Resize_direction::center; break;
```

```
        case 1: expanse_direction = Bmp::Resize_direction::upper_left; break;
        case 2: expanse_direction = Bmp::Resize_direction::upper_right; break;
        case 3: expanse_direction = Bmp::Resize_direction::lower_right; break;
        case 4: expanse_direction = Bmp::Resize_direction::lower_left; break;
    }

    emit ok_button_clicked(ui->vertical_crop->value(),
                           ui->horizontal_crop->value(),
                           expanse_direction, color);

    write_settings();
    close();
}

void Expanse_dialog::write_settings() {
    QSettings settings("My Soft", "LULpainter");

    settings.beginGroup("Expanse dialog");
    settings.setValue("red", color.red());
    settings.setValue("green", color.green());
    settings.setValue("blue", color.blue());
    settings.setValue("vertical_crop", ui->vertical_crop->value());
    settings.setValue("horizontal_crop", ui->horizontal_crop->value());
    settings.setValue("expanse_direction", ui->crop_direction->currentIndex());
    settings.setValue("size", size());
    settings.setValue("pos", pos());
    settings.endGroup();
}

void Expanse_dialog::read_settings() {
    QSettings settings("My Soft", "LULpainter");

    settings.beginGroup("Expanse dialog");
    color.setRgb(qRgb(settings.value("red", 0).toInt(),
                      settings.value("green", 0).toInt(),
                      settings.value("blue", 0).toInt()));
    ui->vertical_crop->setValue(settings.value("vertical_crop", 0).toInt());
    ui->horizontal_crop->setValue(settings.value("horizontal_crop", 0).toInt());
    ui->crop_direction->setCurrentIndex(settings.value("expanse_direction",
0).toInt());
    set_background_color(color);
    resize(settings.value("size", QSize(403, 211)).toSize());
    move(settings.value("pos", QPoint(200, 200)).toPoint());
    settings.endGroup();
}

void Expanse_dialog::on_color_button_clicked() {
    color = QColorDialog::getColor();
    set_background_color(color);
}

void Expanse_dialog::set_background_color(QColor color) {
    ui->color_button->setStyleSheet(QString("background: rgb(") +
QString::number(color.red()) + QString(", ")
                                                               +
QString::number(color.green()) + QString(", ")
                                                               +
QString::number(color.blue()) + QString(");"));
}
```

18. help_dialog.h

```
#include "help_dialog.h"
#include "ui_help_dialog.h"

Help_dialog::Help_dialog(QWidget *parent): QDialog(parent),
                                           ui(new Ui::Help_dialog) {
    ui->setupUi(this);
    setWindowTitle("Help");
}

Help_dialog::~Help_dialog() {
    delete ui;
}
```

19. info_dialog.h

```
#include "info_dialog.h"
Info_dialog::Info_dialog(Bmp_image const* image) {
    setWindowTitle("Image info");
    // here may be memory leaks;
    QHBoxLayout* content = new QHBoxLayout(this);
    QVBoxLayout* labels = new QVBoxLayout(this);
    QVBoxLayout* values = new QVBoxLayout(this);

    QLabel* width = new QLabel("Width:      ", this);
    QLabel* height = new QLabel("Height:     ", this);
    QLabel* size = new QLabel("Size:          ", this);
    QLabel* bitcount = new QLabel("Bitcount: ", this);
    labels->addWidget(width);
    labels->addWidget(height);
    labels->addWidget(size);
    labels->addWidget(bitcount);

    this->width = new QLabel(QString::number(image->get_width()) + QString("
px"), this);
    this->height = new QLabel(QString::number(image->get_height()) + QString("
px"), this);
    this->size = new QLabel(Info_dialog::make_beauty(image->get_size()), this);
    this->bitcount = new QLabel(QString::number(image->get_bitcount()) +
QString(" bits"), this);
    values->addWidget(this->width);
    values->addWidget(this->height);
    values->addWidget(this->size);
    values->addWidget(this->bitcount);

    content->addLayout(labels);
    content->addLayout(values);

    setLayout(content);
}

Info_dialog::~Info_dialog() {
    delete width;
    delete height;
    delete size;
    delete bitcount;
}

void Info_dialog::change_info(Bmp_image const* image) {
    width->setText(QString::number(image->get_width()) + QString(" px"));
    height->setText(QString::number(image->get_height()) + QString(" px"));
```

```
    size->setText(make_beauty(image->get_size()));
    bitcount->setText(QString::number(image->get_bitcount()) + QString("
bits"));
}

QString Info_dialog::make_beauty(int size) {
    std::vector<QString> units{" B", " Kb", " Mb", " Gb"};

    int power = round(log(size)/log(1024));
    power = std::min(power, static_cast<int>(units.size() - 1));
    double beauty_size = size / pow(1024, power);

    QString unit = units.at(power);
    return QString::number(beauty_size, 'g', 3) + unit;
}
```

20. main.cpp

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[]) {
    QApplication app(argc, argv);
    MainWindow main_window;
    main_window.show();

    return app.exec();
}
```

21. mainwindow.h

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
// TODO the highest window title is ".../tiger.bmp" ???? WHAT

MainWindow::MainWindow(QWidget *parent): QMainWindow(parent),
                                         ui(new Ui::MainWindow) {
    ui->setupUi(this);
    setCentralWidget(ui->graphics_view);

    scene = new My_graphics_scene();
    connect(scene, SIGNAL(mouseReleased()), this, SLOT(mouseReleased()));

    prev_file_path = "";
    bmp_image = nullptr;
    grayscale_clicked = false;
    invert_clicked = false;
    cancel_clicked = false;
    changed = false;

    ui->mainToolBar->widgetForAction(ui->actionCrop_3)            -
>setStyleSheet("width: 115px;");
    ui->mainToolBar->widgetForAction(ui->actioncoordinates_gray)  -
>setStyleSheet("width: 115px;");
    ui->mainToolBar->widgetForAction(ui->actioncoordinates_invert)-
>setStyleSheet("width: 115px;");
    ui->mainToolBar->widgetForAction(ui->actionExpanse)           -
>setStyleSheet("width: 115px;");

    read_settings();
}
```

```
MainWindow::~MainWindow() {
    delete bmp_image;
    delete scene;
    delete ui;
}

void MainWindow::mouseReleased() {
    int x1 = scene->get_first().x();
    int y1 = scene->get_first().y();
    int y2 = scene->get_last().y();
    int x2 = scene->get_last().x();

    if (grayscale_clicked) {
        bmp_image->grayscale(x1, y1, x2, y2);
        changed = true;
    } else if (invert_clicked) {
        bmp_image->invert_color(x1, y1, x2, y2);
        changed = true;
    }

    scene->clear();
    scene->addPixmap(QPixmap::fromImage(bmp_image->get_qImage()));
    ui->graphics_view->setScene(scene);
}

void MainWindow::on_actionOpen_triggered() {
    QString file_path = QFileDialog::getOpenFileName(this, "Open a file");
    if (!file_path.isEmpty()) {
        if (changed) {
            Save_dialog* dialog = new Save_dialog(this);
            connect(dialog, SIGNAL(save_button_clicked()), this,
SLOT(on_actionSave_triggered()));
            connect(dialog, SIGNAL(cancel_button_clicked(bool)), this,
SLOT(cancel_toggle()));
            dialog->exec();
            delete dialog;

            if (cancel_clicked) {
                cancel_clicked = false;
                return;
            }
        }

        try {
            Bmp_image* opening_bmp_image = Bmp::bmp(file_path.toStdString());
            if (opening_bmp_image) {
                bmp_image = opening_bmp_image;
                prev_file_path = open_file_path;
                open_file_path = file_path;
                setWindowTitle(QString("LULpainter ") + open_file_path);

                scene->clear();
                delete scene;
                scene = new My_graphics_scene;
                connect(scene, SIGNAL(mouseReleased()), this,
SLOT(mouseReleased()));                                    :(
                scene->addPixmap(QPixmap::fromImage(bmp_image->get_qImage()));
                ui->graphics_view->setScene(scene);

                changed = false;
            } else {
```

```
                QErrorMessage err_msg;
                err_msg.setWindowTitle("Opening file error");
                err_msg.showMessage("Can't open this file on some reason.");
                err_msg.setStyleSheet("QPushButton {"
                                      "    color: white;"
                                      "    background-color: rgb(75, 75, 75);"
                                      "}");
                err_msg.exec();
            }
        } catch (Bmp::Bad_bitcount& err) {
            QErrorMessage err_box;
            err_box.setWindowTitle("Unsupported bitcount");
            err_box.showMessage(QString(err.what()) + QString("\nUploading
image's bitcount: ") + QString::number(err.err_bitcount()));
            err_box.setMaximumSize(490, 170);
            err_box.setMinimumSize(490, 170);
            err_box.setStyleSheet("QPushButton {"
                                  "    color: white;"
                                  "    background-color: rgb(75, 75, 75);"
                                  "}");
            err_box.exec();
        }
    }
}

void MainWindow::on_actionSave_triggered() {
    if (changed) {
        Bmp::save(bmp_image, open_file_path.toStdString());
        changed = false;
    }
}

void MainWindow::on_actionSave_As_triggered() {
    save_as(this);
}

void MainWindow::on_actioncoordinates_gray_triggered() {
    grayscale_toggle();
}

void MainWindow::on_actioncoordinates_invert_triggered() {
    invert_toggle();
}

void MainWindow::on_actionGraysacle_triggered() {
    bmp_image->grayscale();
    scene->clear();
    scene->addPixmap(QPixmap::fromImage(bmp_image->get_qImage()));
    ui->graphics_view->setScene(scene);

    changed = true;
}

void MainWindow::on_actionInvert_triggered() {
    bmp_image->invert_color();
    scene->clear();
    scene->addPixmap(QPixmap::fromImage(bmp_image->get_qImage()));
    ui->graphics_view->setScene(scene);

    changed = true;
}
```

```
void MainWindow::on_actionQuit_triggered() {
    close();
}

void MainWindow::closeEvent(QCloseEvent *event) {
    if (changed) {
        Save_dialog* dialog = new Save_dialog(this);

        if (open_file_path != "")
            connect(dialog, SIGNAL(save_button_clicked(QWidget*)), this,
SLOT(on_actionSave_triggered()));
        else
            connect(dialog, SIGNAL(save_button_clicked(QWidget*)), this,
SLOT(save_as(QWidget*)));

        connect(dialog, SIGNAL(cancel_button_clicked(bool)), this,
SLOT(cancel_toggle()));
        dialog->exec();
        delete dialog;
    }

    if (cancel_clicked) {
        event->ignore();
        cancel_clicked = false;
    } else {
        if (open_file_path == "")
            open_file_path = prev_file_path;

        write_settings();
        event->accept();
    }
}

bool MainWindow::cancel_toggle() {
    return cancel_clicked = true;
}

void MainWindow::on_actionNew_triggered() {
    if (changed) {
        Save_dialog* dialog = new Save_dialog(this);
        connect(dialog, SIGNAL(save_button_clicked(QWidget*)), this,
SLOT(on_actionSave_triggered()));
        connect(dialog, SIGNAL(cancel_button_clicked(bool)), this,
SLOT(cancel_toggle()));
        dialog->exec();
        delete dialog;
    }

    if (cancel_clicked) {
        cancel_clicked = false;
        return;
    }

    scene->clear();

    Bmp_image* opening_bmp_image = Bmp::create(ui->graphics_view->width(), ui-
>graphics_view->height());

    if (opening_bmp_image) {
        delete bmp_image;
        bmp_image = opening_bmp_image;
```

```
        prev_file_path = open_file_path;
        open_file_path = ""; // this is for config
        this->setWindowTitle(QString("LULpainter"));

        scene->addPixmap(QPixmap::fromImage(bmp_image->get_qImage()));
        ui->graphics_view->setScene(scene);

        changed = true;
    }

}

void MainWindow::write_settings() {
    QSettings settings("My Soft", "LULpainter");

    settings.beginGroup("Size and position");
    settings.setValue("size", size());
    settings.setValue("pos", pos());
    settings.endGroup();

    settings.setValue("file_path", open_file_path);
}

void MainWindow::read_settings() {
    QSettings settings("My Soft", "LULpainter");

    settings.beginGroup("Size and position");
    resize(settings.value("size", QSize(400, 400)).toSize());
    move(settings.value("pos", QPoint(200, 200)).toPoint());
    settings.endGroup();

    QString file_path = QString(settings.value("file_path", "").toString());
    if (file_path != "") {
        bmp_image = Bmp::bmp(file_path.toStdString());

        if (bmp_image) {
            open_file_path = file_path;
            this->setWindowTitle(QString("LULpainter ") + open_file_path);

            scene->addPixmap(QPixmap::fromImage(bmp_image->get_qImage()));
            ui->graphics_view->setScene(scene);
        }
    }
}

void MainWindow::grayscale_toggle() {
    ui->mainToolBar->widgetForAction(ui->actioncoordinates_gray)-
>setStyleSheet("width: 115px;"

"background: rgb(150, 150, 150);");
    ui->mainToolBar->widgetForAction(ui->actioncoordinates_invert)-
>setStyleSheet("width: 115px;"

"background: rgb(75, 75, 75);");

    invert_clicked = false;
    grayscale_clicked = true;
}

void MainWindow::invert_toggle() {
```

```
    ui->mainToolBar->widgetForAction(ui->actioncoordinates_gray)-
>setStyleSheet("width: 115px;"

"background: rgb(75, 75, 75);");
    ui->mainToolBar->widgetForAction(ui->actioncoordinates_invert)-
>setStyleSheet("width: 115px;"

"background: rgb(150, 150, 150);");

    invert_clicked = true;
    grayscale_clicked = false;
}

void MainWindow::on_actionCrop_3_triggered() {
    Crop_dialog* crop_dialog = new Crop_dialog(bmp_image->get_width(),
bmp_image->get_height(), this);
    connect(crop_dialog, SIGNAL(ok_button_clicked(int, int,
Bmp::Resize_direction)),
            this,        SLOT(crop_image(int, int, Bmp::Resize_direction)));

    crop_dialog->exec();
    delete crop_dialog;
}

void MainWindow::crop_image(int vertical_crop, int horizontal_crop,
Bmp::Resize_direction crop_direction) {
    bmp_image->crop(vertical_crop, horizontal_crop, crop_direction);

    // some crutch to align center new image                                :(
    delete scene; //                                                        :(
    scene = new My_graphics_scene; //                                       :(
    connect(scene, SIGNAL(mouseReleased()), this, SLOT(mouseReleased())); // :>
    // some crutch to align center new image                                :(

    scene->addPixmap(QPixmap::fromImage(bmp_image->get_qImage()));
    ui->graphics_view->setScene(scene);

    emit image_changed(bmp_image);
    changed = true;
}

void MainWindow::on_actionExpanse_triggered() {
    Expanse_dialog* expanse_dialog = new Expanse_dialog(this);
    connect(expanse_dialog, SIGNAL(ok_button_clicked(int, int,
Bmp::Resize_direction, QColor)),
            this,        SLOT(expanse_image(int, int, Bmp::Resize_direction,
QColor)));

    expanse_dialog->exec();
    delete expanse_dialog;
}

void MainWindow::expanse_image(int vertical_exp, int horizontal_exp,
                               Bmp::Resize_direction expanse_direction, QColor
color) {
    bmp_image->expanse(vertical_exp, horizontal_exp, expanse_direction, color);

    // some crutch to align center new image                                :(
    delete scene; //                                                        :(
    scene = new My_graphics_scene; //                                       :(
    connect(scene, SIGNAL(mouseReleased()), this, SLOT(mouseReleased())); // :>
```

```
    // some crutch to align center new image                                    :(

    scene->addPixmap(QPixmap::fromImage(bmp_image->get_qImage()));
    ui->graphics_view->setScene(scene);

    emit image_changed(bmp_image);
    changed = true;
}

void MainWindow::on_actionImage_info_triggered() {
    Info_dialog* info_dialog = new Info_dialog(bmp_image);
    connect(this, SIGNAL(image_changed(Bmp_image const*)), info_dialog,
SLOT(change_info(Bmp_image const*)));
    info_dialog->setAttribute(Qt::WA_DeleteOnClose);
    info_dialog->setWindowFlags(Qt::WindowStaysOnTopHint);
    info_dialog->show();
}

void MainWindow::on_actionHelp_triggered() {
    Help_dialog* help = new Help_dialog;
    help->setAttribute(Qt::WA_DeleteOnClose);
    help->show();
}

void MainWindow::save_as(QWidget *parent) {
    QString file_path = "";
    if (parent)
        file_path = QFileDialog::getSaveFileName(parent, "Save a file"); // in
case if file dialog will be exec from save_dialog
    else
        file_path = QFileDialog::getSaveFileName(this, "Save a file");

    if (file_path != "") {
        Bmp::save(bmp_image, file_path.toStdString());
        prev_file_path = open_file_path;
        open_file_path = file_path;

        Bmp_image* opening_bmp_image = Bmp::bmp(file_path.toStdString());
        if (opening_bmp_image) {
            bmp_image = opening_bmp_image;
            prev_file_path = open_file_path;
            open_file_path = file_path;
            this->setWindowTitle(QString("LULpainter ") + open_file_path);

            scene->clear();
            // some crutch to align center new
image                                    :(
            delete scene;
//                                                              :(
            scene = new My_graphics_scene;
//                                          :(
            connect(scene, SIGNAL(mouseReleased()), this,
SLOT(mouseReleased())); // :>
            // some crutch to align center new
image                                    :(

            scene->addPixmap(QPixmap::fromImage(bmp_image->get_qImage()));
            ui->graphics_view->setScene(scene);

            changed = false;
        } else {
```

```
            QErrorMessage err_msg;
            err_msg.setWindowTitle("Opening file error");
            err_msg.showMessage("Saving was successfull!"
                                "Can't open this file on some reason.");
            err_msg.setStyleSheet("QPushButton {"
                                  "    color: white;"
                                  "    background-color: rgb(75, 75, 75);"
                                  "}");
            err_msg.exec();
        }
    }
}
```

22. my_graphics_scene.cpp

```
#include "my_graphics_scene.h"
My_graphics_scene::My_graphics_scene(QObject* parent): QGraphicsScene(parent) {}

My_graphics_scene::~My_graphics_scene() {}

void My_graphics_scene::mouseMoveEvent(QGraphicsSceneMouseEvent *event) {
    last = event->scenePos().toPoint();
    emit mouseMoved();
}

void My_graphics_scene::mousePressEvent(QGraphicsSceneMouseEvent *event) {
    first = event->scenePos().toPoint();
    emit mousePressed();
}

void My_graphics_scene::mouseReleaseEvent(QGraphicsSceneMouseEvent *event) {
    last = event->scenePos().toPoint();
    emit mouseReleased();
}

QPoint My_graphics_scene::get_first() {
    return first;
}

QPoint My_graphics_scene::get_last() {
    return last;
}
```

23. save_dialog.cpp

```
#include "save_dialog.h"
#include "ui_save_dialog.h"

Save_dialog::Save_dialog(QWidget *parent): QDialog(parent),
                                           ui(new Ui::save_quit_dialog) {
    ui->setupUi(this);
    this->setModal(true);
}

Save_dialog::~Save_dialog() {
    delete ui;
}

void Save_dialog::on_save_button_clicked() {
    emit save_button_clicked(this);
    done(0);
```

```
}

void Save_dialog::on_cancel_button_clicked() {
    emit cancel_button_clicked(true);
    done(0);
}

void Save_dialog::on_no_button_clicked() {
    done(0);
}

void Save_dialog::closeEvent(QCloseEvent *event) {
    emit cancel_button_clicked(true);
    event->accept();
}
```