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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка изображения в формате PNG

Студент гр. 0382 _____ Ильин Д.А.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург
2021

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Ильин Д.А.

Группа 0382

Тема работы : Обработка изображения в формате PNG

Вариант 11

Программа должна реализовывать следующий функционал по обработке PNG-файла

1. Разделяет изображение на N*M частей. Реализация: либо провести линии заданной толщины, тем самым разделив изображение либо сохранение каждой части в отдельный файл. -- по желанию студента (можно и оба варианта).

Функционал определяется:

- Количество частей по “оси” Y
- Количество частей по “оси” X
- Толщина линии
- Цвет линии
- Либо путь куда сохранить кусочки

2. Рисование прямоугольника. Он определяется:

- Координатами левого верхнего угла
- Координатами правого нижнего угла
- Толщиной линий
- Цветом линий
- Прямоугольник может быть залит или нет
- цветом которым он залит, если пользователем выбран залитый

3. Сделать рамку в виде узора. Рамка определяется:

- Узором (должно быть несколько на выбор. Красивый узор можно получить используя фракталы)
- Шириной
- Цветом

4. Поворот изображения (части) на 90/180/270 градусов. Функционал определяется

- Координатами левого верхнего угла области
- Координатами правого нижнего угла области
- Углом поворота

Дата выдачи задания: 25.04.2021

Дата сдачи реферата: 31.05.2021

Дата защиты реферата: 31.06.2021

Студент _____ Ильин Д.А.

Преподаватель _____ Берленко Т.А.

АННОТАЦИЯ

В процессе выполнения курсовой работы, были созданы соответствующие функции для обработки изображения формата PNG, типа цвета RGBA. Была использована библиотека libpng, для создания графического интерфейса был использован фреймворк Qt.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель курсовой работы заключается в создание программы , для обработки изображения формата PNG с типом цвета PNG. В ходе работы были задействованы язык C++, библиотека libpng, среда разработки Qt.

1.ЦЕЛЬ И ЗАДАНИЕ РАБОТЫ

1.1 Цель работы.

Целью данной программы является создание программы обрабатывающей изображение формата PNG. Освоить работу с библиотекой libpng. Научится применять язык программирования C++.

1.2 Задание.

Вариант 11

Программа должна реализовывать следующий функционал по обработке PNG-файла

1. Разделяет изображение на N*M частей. Реализация: либо провести линии заданной толщины, тем самым разделив изображение либо сохранение каждой части в отдельный файл. -- по желанию студента (можно и оба варианта).

Функционал определяется:

- Количество частей по “оси” Y
- Количество частей по “оси” X
- Толщина линии
- Цвет линии
- Либо путь куда сохранить кусочки

2. Рисование прямоугольника. Он определяется:

- Координатами левого верхнего угла
- Координатами правого нижнего угла
- Толщиной линий
- Цветом линий
- Прямоугольник может быть залит или нет
- цветом которым он залит, если пользователем выбран залитый

3. Сделать рамку в виде узора. Рамка определяется:

- Узором (должно быть несколько на выбор. Красивый узор можно получить используя фракталы)
- Шириной
- Цветом

4. Поворот изображения (части) на 90/180/270 градусов. Функционал определяется

- Координатами левого верхнего угла области
- Координатами правого нижнего угла области
- Углом поворота

2.ХОД РАБОТЫ

2.1. Чтение/запись изображения

Для работы с изображением был написан класс *Image_PNG*. Для чтения используется метод *readFromDisk()* который принимает путь к файлу считывает изображения с помощью средств библиотеки *libpng* , и возвращает в случае удачи считывания 1, иначе 0. Для записи используется метод *writeOnDisk()* которая , если не передать путь , сохраняет изображение в тот же файл с которого считало, если передать путь записывает изображение на диск по этому пути. Функция также использует функции *libpng* для записи .

2.2.Решение подзадачи 1

В данной подзадаче нужно было разрезать изображение N*M частей, N и M, цвет и ширину линии разбиения вводит пользователь. Реализация исполнена с помощью метода *drawLineNM(),* который в свою очередь использует функцию *drawLine()***,** которая рисует линию определённой толщины и цвета по введенным координатам.

2.2.Решение подзадачи 2

Во второй задаче требовалось нарисовать прямоугольник по введённым координатам, с возможностью выбора толщины, цвета , а также заливки. Реализация этой задачи находится в функции *drawRectangle()*, которая использует функцию *drawLine()*, которая рисует линию определённой толщины и цвета по введенным координатам.

2.3.Решение подзадачи 3

Здесь нужно было нарисовать рамку вокруг изображения, причём вариаций рамок должно быть несколько. Для реализации использовалась функция *drawRectangle(),* для рисования прямоугольника определённого цвета и толщины, вокруг всей картинки.

2.4.Решение подзадачи 4

Для поворота изображения (или его части) были использованы функции: *turnImagePis90(), turnImagePis180(), turnImagePis270(), turnImage90(), turnImage180()*. Каждая из которых поворачивает изображение или же его часть на соответствующий угол.

2.5.Реализация GUI

Для создания GUI был использован фреймворк Qt. Для ввода пользователем данных были использованы диалоговые окна, при вводе неверных данных в которые пользователь получит соответствующее сообщение об ошибке. Также была реализована справка о приложении.

ЗАКЛЮЧЕНИЕ

В ходе работы была создана программа удовлетворяющая поставленным требованиям. Для взаимодействия с пользователем был создан GUI.

КОД ПРОГРАММЫ

Файл **images.h**:

```
#ifndef IMAGE_H
#define IMAGE_H

#include <png.h>
#include <QString>
#include <QVector>
#include <QColor>
#include <QString>
```

```
const int SIZE_BIT_CHECK_PNG = 8;

enum RGBA{//канал
    R, //=0
    G, //=1
    B, //=2
    A  //=3
};

/* объявление класса*/
class Images {
private:
    int m_width, m_height;//ширина и высота в изб. в пикселях
    png_byte **m_arr_pixel = NULL;//масив пикселей  изображение (трех
мерный массив)
    png_byte m_color_type; //тип кодирования цвета (может быть только
RGBA)
    png_byte m_bit_depth;  //бит глубины
    QString m_path_to_img = "";// путь к изоб. (включая имя файла) // ""
- означает что структура не инициализированна.

    png_structp m_png_ptr; //по факту изначально само изображение
    png_infop m_info_ptr; //информация об изображении
    int m_number_of_passes;

    void set_first_small_second_big(int *small_p, int *big_p);//в первую
переменую устанавливает меньшее,
                                                              //во второую
большее значение из переданых переменых
    void freeImage();   //вспомогательная фнкция для деструктора
    void deepCopy(const Images &img);    //вспомогательная фнкция для
конструктора копирования и operator=
    void setColorInPixel(int x, int y, QColor color);  //устанавливает в
пиксель цвет //если кординаты вне картинки ничего не делает
    QColor getColorInPixel(int x, int y); //получает цвет из пикселя
public:
    //Конструктор
    Images();

    ~Images();//деконструктор

    int readFromDisk(QString path); //считывание изб. с диска

    bool wasInitialized(); //определяет была ли инициализирована
структура

    int getHeight();//получить высоту изб. в пикс.

    int getWidth();//получить ширину в пикс.

    QString getPath();//получить потуть

    void writeOnDisk(); // записываем изб. на диск

    //нарисовать отрезок
    void drawLine(int x1, int y1, int x2, int y2,int d , QColor color);

    //нарисовать прямоугольник //если color_in isValid то ресуется
залитый
```

```
    void drawRectangle(int x1, int x2,int y1,int y2,int d ,QColor
color_out = QColor::Invalid,  QColor color_in = QColor::Invalid);

    //разделение изображения на части
    void drawLineNM(int n, int m, int d, QColor color);

    //поворот изображения
    void turnImagePis90(int x1, int y1, int x2, int y2);

    //поворот изображения
    void turnImagePis180(int x1, int y1, int x2, int y2);

    //поворот изображения
    void turnImagePis270(int x1, int y1, int x2, int y2);

    //поворот изображения
    void turnImage90();

    //поворот изображения
    void turnImage180();

};

#endif // IMAGE_H
```

Файл **images.cpp**:

```
#include "images.h"

#include <QMessageBox>
#include <QFile>
#include <QtMath>

void Images::writeOnDisk(){
    std::string tmp_str =   m_path_to_img.toStdString();
    const char *c_path = tmp_str.c_str();

    /* Открываем файл для бин. чтения*/
    FILE *fp = fopen(c_path, "wb");
    if (fp == NULL){
        // Some error handling: file could not be opened
        QMessageBox::critical(nullptr,"Ошибка","Файл с таким именем не
найден");
        return ;
    }

    /* Выделение дин. памяти */
    m_png_ptr = png_create_write_struct(PNG_LIBPNG_VER_STRING, NULL,
NULL, NULL);
```

```
    if (m_png_ptr == NULL){
        fclose(fp);
        // Some error handling: png_create_write_struct failed

        QMessageBox::critical(nullptr,"Ошибка","Не удалось выделить дин.
память для структуры png_ptr");
        return ;
    }

    m_info_ptr = png_create_info_struct(m_png_ptr);
    if (m_info_ptr == NULL){
        png_destroy_write_struct(&m_png_ptr, NULL);
        fclose(fp);
        // Some error handling: png_create_info_struct failed

        QMessageBox::critical(nullptr,"Ошибка","Не удалось выделить дин.
память для структуры info_ptr");
        return ;
    }

    if (setjmp(png_jmpbuf(m_png_ptr))){
        png_destroy_write_struct(&m_png_ptr, &m_info_ptr);
        fclose(fp);
        // Some error handling: error during init_io

        QMessageBox::critical(nullptr,"Ошибка","Произошла ошибка при
записи информации о файле");
        return ;
    }

    png_init_io(m_png_ptr, fp);

    /* write header */
    if (setjmp(png_jmpbuf(m_png_ptr))){
        png_destroy_write_struct(&m_png_ptr, &m_info_ptr);
        fclose(fp);
        // Some error handling: error during writing header
    }

    png_set_IHDR(m_png_ptr, m_info_ptr, m_width, m_height,
                 m_bit_depth, m_color_type, PNG_INTERLACE_NONE,
                 PNG_COMPRESSION_TYPE_BASE, PNG_FILTER_TYPE_BASE);

    png_write_info(m_png_ptr, m_info_ptr);

    /* write bytes */
    if (setjmp(png_jmpbuf(m_png_ptr))){
        png_destroy_write_struct(&m_png_ptr, &m_info_ptr);
        fclose(fp);
        // Some error handling: error during writing bytes

        QMessageBox::critical(nullptr,"Ошибка","Произошла ошибка при
записи изображения");
        return ;
    }

    png_write_image(m_png_ptr, m_arr_pixel);

    /* end write */
```

```
    if (setjmp(png_jmpbuf(m_png_ptr))){
        png_destroy_write_struct(&m_png_ptr, &m_info_ptr);
        fclose(fp);
        // Some error handling: error during end of write

        QMessageBox::critical(nullptr,"Ошибка","Произошла ошибка при
записи конца файла");
        return ;
    }

    png_write_end(m_png_ptr, NULL);

    png_destroy_write_struct(&m_png_ptr, &m_info_ptr);
    fclose(fp);
}

bool Images::wasInitialized(){
    if (m_path_to_img == ""){
        return false;
    }else {
        return true;
    }
}

void  Images::freeImage(){
    if (m_arr_pixel != NULL){
        for (int i = 0; i < m_height; i++)
            free(m_arr_pixel[i]);
        free(m_arr_pixel);
    }
    m_arr_pixel = NULL;
}

//Деструктор
Images::~Images() {
    freeImage();
}

//Конструктор
Images::Images() {    }

//чтение изображения с диска
int Images::readFromDisk( QString path){
    char header[SIZE_BIT_CHECK_PNG];    // 8 is the maximum size that can
be checked

    std::string tmp_str = path.toStdString();
    const char *c_path = tmp_str.c_str();

    // open file and test for it being a png
    FILE *fp = fopen(c_path, "rb");
    if (fp == NULL){
        // Some error handling: file could not be opened
        QMessageBox::critical(nullptr,"Ошибка","Файл с таким именем не
найден");
        return 1;
    }
```

```
    fread(header,sizeof (header[0]), SIZE_BIT_CHECK_PNG, fp);//считывает
массив эл из файла
    if (png_sig_cmp((png_const_bytep)header, 0, SIZE_BIT_CHECK_PNG))
{//проверка что файл это png
        fclose(fp);
        // Some error handling: file is not recognized as a PNG

        QMessageBox::critical(nullptr,"Ошибка","Этот файл не PNG\n\
nДанная прогрмма потдерживает только файлы формата *.png, с типом цвета
RGBA");
        return 1;
    }

    // выделение памяти для структур
    m_png_ptr = png_create_read_struct(PNG_LIBPNG_VER_STRING, NULL, NULL,
NULL);
    if (m_png_ptr == NULL){
        fclose(fp);
        // Some error handling: png_create_read_struct failed
        QMessageBox::critical(nullptr,"Ошибка","Не удалось выделить дин.
память для структуры png_ptr");
        return 1;
    }

    m_info_ptr = png_create_info_struct(m_png_ptr);
    if (m_info_ptr == NULL){
        png_destroy_read_struct(&m_png_ptr,NULL, NULL);//очистка массива
        fclose(fp);
        // Some error handling: png_create_info_struct failed

        QMessageBox::critical(nullptr,"Ошибка","Не удалось выделить дин.
память для структуры info_ptr");
        return 1;
    }

    if (setjmp(png_jmpbuf(m_png_ptr))){//обработка ошибок
        png_destroy_read_struct(&m_png_ptr,&m_info_ptr, NULL);//очистка
массива
        fclose(fp);
        // Some error handling: error during init_io

        QMessageBox::critical(nullptr,"Ошибка","Не удалось считать данные
об изображении");
        return 1;
    }

    png_init_io(m_png_ptr, fp); //настройка функция ввода/вывода
    png_set_sig_bytes(m_png_ptr, 8); //сообщить библиотеке что первые 8
байт отстутствуют

    png_read_info(m_png_ptr, m_info_ptr);//считывание файла с изб. в
структуру

    //проверяем что файл png RGBA
    if (png_get_color_type(m_png_ptr, m_info_ptr) !=
PNG_COLOR_TYPE_RGB_ALPHA){
        png_destroy_read_struct(&m_png_ptr, &m_info_ptr, NULL);//очистка
структуры
        fclose(fp);
```

```
        QMessageBox::critical(nullptr,"Ошибка","У этого png файла тип
цвета не RGBA\n\nДанная программа работает только с файлами png  , с
типом цвета  RGBA!!!");
        return 1;
    }

    //очищаем память от прошлого изображения(если оно было)
    freeImage();

    m_width = png_get_image_width(m_png_ptr, m_info_ptr);
    m_height = png_get_image_height(m_png_ptr, m_info_ptr);
    m_color_type = png_get_color_type(m_png_ptr, m_info_ptr);
    m_bit_depth = png_get_bit_depth(m_png_ptr, m_info_ptr);

    m_number_of_passes = png_set_interlace_handling(m_png_ptr);
    png_read_update_info(m_png_ptr, m_info_ptr);

    // read file
    if (setjmp(png_jmpbuf(m_png_ptr))){
        png_destroy_read_struct(&m_png_ptr, &m_info_ptr, NULL);//очистка
структуры
        fclose(fp);
        // Some error handling: error during read_image

        QMessageBox::critical(nullptr,"Ошибка","Не удалось считать само
изображение");
        return 1;
    }

    // выделение памяти
    m_arr_pixel = (png_bytep *) malloc(sizeof(png_bytep) * m_height);
    for (int i = 0; i < m_height; i++)
        m_arr_pixel[i] = (png_byte *) malloc(png_get_rowbytes(m_png_ptr,
m_info_ptr));

    // наконец считывание изображения
    png_read_image(m_png_ptr, m_arr_pixel);

    png_destroy_read_struct(&m_png_ptr, &m_info_ptr, NULL);//очистка
структуры
    fclose(fp);

    m_path_to_img = path;
    return 0;

}

int Images::getHeight() {
    return m_height;
}

int Images::getWidth() {
    return m_width;
}

void Images::drawLine(int x1, int y1, int x2, int y2,int d , QColor color
) {
    int x,y;
```

```
    if (x1 == x2){ //вертикальная линия (tg равен inf)
        //уравнение вида x == const

        int inc_left = trunc((float)(d-1)/2);//окр в меньшую
        int inc_right = round((float)(d-1)/2);//окр в большую

        set_first_small_second_big(&y1,&y2);
        for (y = y1; y <= y2; y++){
            for(x = x1 - inc_left; x <= x1+inc_right; x++){ //создание
тольщины линии
                setColorInPixel(x, y, color);
            }
        }
        return;
    }

    float tg_a = (float)(y1 - y2)/(x1 - x2);
    float b = y1 - tg_a * x1;

    if (abs(tg_a) <= 1) {   //тогда строим y = y(x)
        set_first_small_second_big(&x1,&x2);

        int inc_up = trunc((float)(d-1)/2);//окр в меньшую
        int inc_down = round((float)(d-1)/2);//окр в большую
        int y_main;

        for ( x = x1; x <= x2; x++) {
            y_main = round(tg_a*x + b);  // y = kx+b

            for(y = y_main - inc_up; y <= y_main+inc_down; y++){
//создание тольщины линии
                setColorInPixel(x, y, color);
            }
        }

    } else { //строим x = x(y)
        set_first_small_second_big(&y1,&y2);

        int inc_left = trunc((float)(d-1)/2);//окр в меньшую
        int inc_right = round((float)(d-1)/2);//окр в большую
        int x_main;

        for ( y = y1; y <= y2; y++) {
            x_main = round( (y - b) / tg_a );  // x = (y-b)/k

            for(x = x_main - inc_left; x <= x_main+inc_right; x++){
//создание тольщины линии
                setColorInPixel(x, y, color);
            }
        }
    }
}

void Images::set_first_small_second_big(int *small_p, int *big_p){
    int small = *small_p;
    int big = *big_p;

    if (small > big) {//нужно наоборот
        *small_p = big;
```

```
            *big_p = small;
        }
    }

    void Images::setColorInPixel(int x, int y,QColor color) {
        png_byte *pixel;

        if (x >= 0 && x < m_width && y >= 0 && y < m_height){
            pixel = &(m_arr_pixel[y][x*4]);
            pixel[R] = color.red();
            pixel[G] = color.green();
            pixel[B] = color.blue();
            pixel[A] = color.alpha();
        }
    }

    QColor Images::getColorInPixel(int x, int y) {
        png_byte *pixel;
        QColor color;

        if (x >= 0 && x < m_width && y >= 0 && y < m_height){
            pixel = &(m_arr_pixel[y][x*4]);
            color.setRed(pixel[R]);
            color.setRed(pixel[G]);
            color.setRed(pixel[B]);
            color.setRed(pixel[A]);
        }
        return color;
    }

    QString Images::getPath(){
        return m_path_to_img;
    }

    void Images::turnImagePis180(int x1, int y1, int x2, int y2){
        set_first_small_second_big(&x1, &x2);
        set_first_small_second_big(&y1, &y2);
        int width = x2 - x1;
        int height = y2 - y1;
        png_bytep * new_arr_pix = (png_bytep *) malloc(sizeof(png_bytep) *
    m_height);
        for(int i = 0; i < m_height; i++){
            new_arr_pix[i] = (png_bytep) malloc(4*sizeof(png_byte)*m_width);
        }
        QColor color;

        for (int y = 0; y < m_height; y++){
            for (int x = 0; x < m_width; x++){
                new_arr_pix[y][x*4 + R] = m_arr_pixel[y][x*4 + R];
                new_arr_pix[y][x*4 + G] = m_arr_pixel[y][x*4 + G];
                new_arr_pix[y][x*4 + B] = m_arr_pixel[y][x*4 + B];
                new_arr_pix[y][x*4 + A] = m_arr_pixel[y][x*4 + A];

            }
        }

        for (int x = 0; x < width; x++){
            for (int y = 0; y < height; y++){
                color.setRed(new_arr_pix[y2 - y - 1][(x2-x)*4 + R]);
```

```
            color.setGreen(new_arr_pix[y2 - y - 1][(x2-x)*4 + G]);
            color.setBlue(new_arr_pix[y2 - y - 1][(x2-x)*4 + B]);
            color.setAlpha(new_arr_pix[y2 - y - 1][(x2-x)*4 + A]);
            setColorInPixel(x1+x, y + y1, color);
        }
    }
    for(int i = 0; i < m_height; i++){
        free(new_arr_pix[i]);
    }
    free(new_arr_pix);
}

void Images::turnImagePis90(int x1, int y1, int x2, int y2){
    set_first_small_second_big(&x1, &x2);
    set_first_small_second_big(&y1, &y2);
    int height = y2 - y1;
    png_bytep * new_arr_pix = (png_bytep *) malloc(sizeof(png_bytep) *
m_height);
    for(int i = 0; i < m_height; i++){
        new_arr_pix[i] = (png_bytep) malloc(4*sizeof(png_byte)*m_width);
    }
    QColor color;

    for (int y = 0; y < m_height; y++){
        for (int x = 0; x < m_width; x++){
            new_arr_pix[y][x*4 + R] = m_arr_pixel[y][x*4 + R];
            new_arr_pix[y][x*4 + G] = m_arr_pixel[y][x*4 + G];
            new_arr_pix[y][x*4 + B] = m_arr_pixel[y][x*4 + B];
            new_arr_pix[y][x*4 + A] = m_arr_pixel[y][x*4 + A];

        }
    }

    for (int x = x1; x < x2; x++){
        for (int y = 0; y < height; y++){
            color.setRed(new_arr_pix[y2 - y - 1][x*4 + R]);
            color.setGreen(new_arr_pix[y2 - y - 1][x*4 + G]);
            color.setBlue(new_arr_pix[y2 - y - 1][x*4 + B]);
            color.setAlpha(new_arr_pix[y2 - y - 1][x*4 + A]);
            setColorInPixel(y + y1, x, color);
        }
    }
    for(int i = 0; i < m_height; i++){
        free(new_arr_pix[i]);
    }
    free(new_arr_pix);
}

void Images::turnImagePis270(int x1, int y1, int x2, int y2){
    set_first_small_second_big(&x1, &x2);
    set_first_small_second_big(&y1, &y2);
    int width = x2 - x1;
    png_bytep * new_arr_pix = (png_bytep *) malloc(sizeof(png_bytep) *
m_height);
    for(int i = 0; i < m_height; i++){
        new_arr_pix[i] = (png_bytep) malloc(4*sizeof(png_byte)*m_width);
    }
    QColor color;

    for (int y = 0; y < m_height; y++){
```

```
            for (int x = 0; x < m_width; x++){
                new_arr_pix[y][x*4 + R] = m_arr_pixel[y][x*4 + R];
                new_arr_pix[y][x*4 + G] = m_arr_pixel[y][x*4 + G];
                new_arr_pix[y][x*4 + B] = m_arr_pixel[y][x*4 + B];
                new_arr_pix[y][x*4 + A] = m_arr_pixel[y][x*4 + A];

            }
        }

        for (int x = 0; x < width; x++){
            for (int y = y1; y < y2; y++){
                color.setRed(new_arr_pix[y][(x2 - x - 1)*4 + R]);
                color.setGreen(new_arr_pix[y][(x2 - x - 1)*4 + G]);
                color.setBlue(new_arr_pix[y][(x2 - x - 1)*4 + B]);
                color.setAlpha(new_arr_pix[y][(x2 - x - 1)*4 + A]);
                setColorInPixel(y, x + x1, color);
            }
        }
        for(int i = 0; i < m_height; i++){
            free(new_arr_pix[i]);
        }
        free(new_arr_pix);
    }

    void Images::drawLineNM(int n, int m, int d, QColor color){
        int x, y;
        float shift_x = m_width/n;
        float shift_y = m_height/m;
        for(x = shift_x; x < m_width; x += shift_x){
            drawLine(trunc(x), 0, trunc(x), m_height, d, color);
        }
        for(y = shift_y; y < m_height; y += shift_y){
            drawLine(0, trunc(y), m_width, trunc(y), d, color);
        }
    }

    void Images::turnImage90()
    {
        int help;
        png_bytep * new_arr_pix = (png_bytep *) malloc(sizeof(png_bytep) *
    m_width);
        for(int i = 0; i < m_width; i++){
            new_arr_pix[i] = (png_bytep) malloc(4*sizeof
    (png_byte)*m_height);
        }

        int new_height = m_width;
        int new_width = m_height;

        for (int y = 0; y < new_height; y++){
            for (int x = 0; x < new_width; x++){
                new_arr_pix[y][x*4 + R] = m_arr_pixel[m_height - x - 1][y*4 +
    R];
                new_arr_pix[y][x*4 + G] = m_arr_pixel[m_height - x - 1][y*4 +
    G];
                new_arr_pix[y][x*4 + B] = m_arr_pixel[m_height - x - 1][y*4 +
    B];
                new_arr_pix[y][x*4 + A] = m_arr_pixel[m_height - x - 1][y*4 +
    A];
```

```
        }
    }
    m_arr_pixel = new_arr_pix;
    help = m_height;
    m_height = m_width;
    m_width = help;
}

void Images::drawRectangle(int x1, int x2,int y1,int y2,int d ,QColor
color_out,  QColor color_in ){
    set_first_small_second_big(&x1, &x2);
    set_first_small_second_big(&y1, &y2);
    int x = x1;
    int y = y1;
    int width = x2 - x1;
    int height = y2 - y1;

    int inc_left = (d-1)/2;//окр в меньшую
    int inc_right = round((float)(d-1)/2);//окр в большую
    int inc_up = inc_left;//окр в меньшую
    int inc_down = inc_right;//окр в большую

    if (color_out.isValid()) {
        drawLine(x , y + inc_up, x + width-1, y + inc_up, d, color_out);
// up ---
        drawLine(x+width-inc_right-1, y+d, x+width-inc_right-1, y +
height - d-1, d, color_out);    // right |
        drawLine(x, y+height-inc_down-1, x + width-1, y+height-inc_down-
1, d, color_out);   // down ---
        drawLine(x+inc_left, y+d, x+inc_left, y + height - d-1, d,
color_out);    // left |
    }

    if (color_in.isValid()) {
        for(int i = x+d; i < x+width-d; i++) {
            for (int j = y+d; j < y+height-d;j++ ) {
                setColorInPixel(i,j,color_in);
            }
        }
    }
}
```

Файл **main.cpp**:

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

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```