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

# КУРСОВАЯ РАБОТА

**по дисциплине «Программирование»**

**Тема: Обработка изображений**

Студент гр. 0382 _____ Шангичев В. А.

Преподаватель _____ Берленко Т. А.

Санкт-Петербург

2021

**ЗАДАНИЕ**
**НА КУРСОВУЮ РАБОТУ**

Студент Шангичев В. А.

Группа 0382

Тема работы: Обработка изображений в формате PNG

Исходные данные:
Вариант 16.
Программа должна реализовывать следующий функционал по обработке PNG-файла

1. Копирование заданной области. Функционал определяется:
    - Координатами левого верхнего угла области-источника
    - Координатами правого нижнего угла области-источника
    - Координатами левого верхнего угла области-назначения
2. Заменяет все пиксели одного заданного цвета на другой цвет. Функционал определяется:
    - Цвет, который требуется заменить
    - Цвет на который требуется заменить
3. Сделать рамку в виде узора. Рамка определяется:
    - Узором (должно быть несколько на выбор. Красивый узор можно получить используя фракталы)
    - Цветом
    - Шириной
4. Поиск всех залитых прямоугольников заданного цвета. Требуется найти все прямоугольники заданного цвета и обвести их линией. Функционал определяется:
    - Цветом искомых прямоугольников
    - Цветом линии для обводки

- Толщиной линии для обводки

Дата выдачи задания: 25.04.2021

Дата сдачи реферата: 19.05.2021

Дата защиты реферата: 21.05.2021

Студент _____ Шангичев В. А.

Преподаватель _____ Берленко Т. А.

# АННОТАЦИЯ

В ходе выполнения данной курсовой работы была создана программа для обработки изображений типа PNG с форматом пикселей RGB. Для считывания, записи и изменения содержимого файлов использовалась библиотека libpng. Также с помощью фреймворка Qt был создан графический интерфейс.

**СОДЕРЖАНИЕ**

# ВВЕДЕНИЕ

В данной работе требовалось написать программу, обрабатывающую изображения в формате PNG с форматом пикселей RGB. Для реализации данной программы использовался язык C++, библиотека libpng и фреймворк Qt.

# 1. ЦЕЛЬ И ЗАДАНИЕ РАБОТЫ

## 1.1. Цель работы.

Целью данной работы является разработка программы, обрабатывающей изображения.

## 1.2 Задание.

Вариант 16.

Программа должна реализовывать следующий функционал по обработке PNG-файла

1. Копирование заданной области. Функционал определяется:
    - Координатами левого верхнего угла области-источника
    - Координатами правого нижнего угла области-источника
    - Координатами левого верхнего угла области-назначения
2. Заменяет все пиксели одного заданного цвета на другой цвет. Функционал определяется:
    - Цвет, который требуется заменить
    - Цвет на который требуется заменить
3. Сделать рамку в виде узора. Рамка определяется:
    - Узором (должно быть несколько на выбор. Красивый узор можно получить используя фракталы)
    - Цветом
    - Шириной
4. Поиск всех залитых прямоугольников заданного цвета. Требуется найти все прямоугольники заданного цвета и обвести их линией. Функционал определяется:
    - Цветом искомых прямоугольников
    - Цветом линии для обводки
    - Толщиной линии для обводки

# 2. ХОД ВЫПОЛНЕНИЯ РАБОТЫ

## 2.1. Чтение / запись изображения

Для чтения и записи изображения были разработаны функции `read_png_file` и `write_png_file` соответственно. Также была создана структура `Png` для удобного представления обрабатываемого файла.

## 2.2 Решение подзадачи 1.

Для копирования заданной области изображения была создана функция `copy_frame` и структура `FrameCopy`, содержащая в себе координаты левого верхнего и правого нижнего угла копируемой области и координаты левого верхнего угла области назначения. Хранение координат реализовано с помощью структуры `Edge`.

В функции `copy_frame` сначала вычисляется длина и ширина фрагмента, который необходимо скопировать. Далее создается двумерный массив, в который записываются те пиксели изображения, которые необходимо скопировать. Для копирования значений каналов одного пикселя используется вспомогательная функция `replace`, по назначению напоминающая `memcpy`.

Далее значения этих пикселей заносятся в соответствующую область на изображении, и производится освобождение памяти.

## 2.3 Решение подзадачи 2.

Для замены одного цвета на другой была реализована функция `change_color`, принимающая на вход структуру `Png`, цвет, который необходимо заменить и цвет, на который надо заменять соответствующие пиксели. В функции через циклы производится перебор всех пикселей, их сравнение с переданным цветом с помощью функции `equal`, и если цвета совпали, то происходит замена цвета с помощью функции `replace`.

### 2.4 Решение подзадачи 3.

Следующим этапом работы является создание рамки. Рамка может быть двуцветной или одноцветной. Реализовывается данная опция путем создания функции `get_ornament`. В функции резервируется память для хранения измененного изображения, после чего туда копируются те пиксели изображения, которые останутся неизменными. Далее применяется троекратный вызов функции `get_frame`, производящей “обводку” заданной области определенным цветом.

### 2.4 Решение подзадачи 4.

Для решения данной подзадачи используется функция `find_rectangles` и вспомогательные функции `search_rectangle`, `find_right`, `find_diagonal`, `find_down` и другие. Алгоритм нахождения прямоугольника следующий: как только будет найден пиксель заданного цвета, не принадлежащий к какому-либо уже обработанному прямоугольнику, с помощью функции `search_rectangle` находится прямоугольник, левым верхним углом которого является найденный пиксель. Далее с помощью использовавшейся ранее функции `get_frame` найденные прямоугольники обводятся.

### 2.6 Создание GUI

Для создания GUI использовался фреймворк Qt. Для ввода данных пользователем были созданы диалоговые окна. В случае ошибки пользователь получает соответствующие сообщения. Также была реализована справка, выводящая окно с информацией о созданном приложении.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения данного задания была создана программа для обработки изображений. Для взаимодействия с пользователем был создан GUI с помощью фреймворка Qt.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


1. ***Документация Qt:*** ***https://www.qt.io/***
2. ***Сайт вопросов:*** https://stackoverflow.com/

# ПРИЛОЖЕНИЕ А
# ИСХОДНЫЙ КОД

Файл src/course_work/copy_frame.h

```
#ifndef COPY_FRAME_H
#define COPY_FRAME_H
#include "png_structs.h"

void copy_frame(struct Png* image, FrameCopy coords);

#endif // COPY_FRAME_H
```

Файл src/course_work/dialog.h

```
#ifndef DIALOG_H
#define DIALOG_H

#include <QDialog>
#include "png_structs.h"
#include <QLineEdit>
#include <QIntValidator>
#include <QDebug>

namespace Ui {
class Dialog;
}

class Dialog : public QDialog
{
    Q_OBJECT

public:
    explicit Dialog(int width, int height, QWidget *parent = nullptr);
    ~Dialog();
    void set_frame(FrameCopy* fcopy, int* wclosed){
        frame_copy = fcopy;
        was_closed = wclosed;
        *(was_closed) = 1;
    }
```

```
private slots:
    void on_pushButton_clicked();
    void onTextchanged(const QString & text);
    int check();
    void clean(QLineEdit* obj);

private:
    Ui::Dialog *ui;
    struct FrameCopy* frame_copy;
    int* was_closed;
};

#endif // DIALOG_H
```

Файл src/course_work/get_ornament.h

```
#ifndef GET_ORNAMENT_H
#define GET_ORNAMENT_H
#include "png_structs.h"

void get_ornament(struct Png* image, int ornament_width, png_byte* color,
png_byte* second_color);
void get_frame(struct Png* image, png_byte* line_color, int line_width,
Rectangle rectangle);

#endif // GET_ORNAMENT_H
```

Файл src/course_work/imageviewer.h

```
#ifndef IMAGEVIEWER_H
#define IMAGEVIEWER_H
#include "png_structs.h"
#include <QMainWindow>
#include <QGraphicsScene>
#include <QColorDialog>
#include <QInputDialog>
#include <QMessageBox>
#include "dialog.h"
#include "read_write_functions.h"
```

```
#include "pixel_functions.h"
#include "get_ornament.h"
#include "search_rectangles.h"
#include "copy_frame.h"

namespace Ui {
class ImageViewer;
}

class ImageViewer : public QMainWindow
{
    Q_OBJECT

public:
    explicit ImageViewer(QWidget *parent = nullptr);
    ~ImageViewer();
    void start();
    void setFile(QString filepath){
        filename = filepath;
    }

    void init(){
        scene = new QGraphicsScene(this);
        png_file = new struct Png;
    }



private slots:
    void set_ornament();
    void refresh();
    void on_actionsearch_rectangles_triggered();
    int get_color(png_byte color[], QString info);
    void on_actionchange_color_triggered();
    void on_actioncopy_frame_triggered();
    int valid(struct FrameCopy* fcopy);

private:
```

```
    Ui::ImageViewer *ui;
    QGraphicsScene* scene;
    QPixmap image;
    struct Png* png_file;
    QString filename;
};

#endif // IMAGEVIEWER_H
```

Файл src/course_work/mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include "dialog.h"
#include "png_structs.h"
#include <QMainWindow>
#include <QFileDialog>
#include <QMessageBox>


QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void on_pushButton_clicked();

    void on_pushButton_2_clicked();

private:
```

```
    Ui::MainWindow *ui;
};
#endif // MAINWINDOW_H
```

Файл src/course_work/pixel_functions.h

```
#ifndef PIXEL_FUNCTIONS_H
#define PIXEL_FUNCTIONS_H
#include "png_structs.h"

int equal(png_byte* image_pixel, png_byte* color, int width);
void replace(png_byte* image_pixel, png_byte* color, int width);
void change_color(struct Png* image, png_byte* replaced_color, png_byte*
color_to_replace);
int in(int y, int x, Rectangle rectangle);
int checked(int y, int x, Rectangle* rectangles, int len);

#endif // PIXEL_FUNCTIONS_H
```

Файл src/course_work/png_structs.h

```
#ifndef PNG_STRUCTS_H
#define PNG_STRUCTS_H
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdarg.h>
#include "png.h"


typedef struct Edge{
    /* this struct determines coordinates
       of edge */
    int x, y;
} Edge;

typedef struct FrameCopy{
    /* struct for coping frame */
    Edge left_top;
    Edge right_bottom;
```

```
    Edge destination;
} FrameCopy;

typedef struct Rectangle{
    Edge left_top;
    Edge right_bottom;
} Rectangle;

struct Png{
    int width, height;
    png_byte color_type;
    png_byte bit_depth;

    png_structp png_ptr;
    png_infop info_ptr;
    int number_of_passes;
    png_bytep *row_pointers;
    int pixel_width;
};
#endif // PNG_STRUCTS_H
```

Файл src/course_work/read_write_functions.h

```
#ifndef READ_WRITE_FUNCTIONS_H
#define READ_WRITE_FUNCTIONS_H
#include "png_structs.h"
#include <QString>
#include <QMainWindow>

void write_png_file(struct Png* image, QString file_name);
int read_png_file(struct Png* image, QString filename, QMainWindow* win);

#endif // READ_WRITE_FUNCTIONS_H
```

Файл src/course_work/search_rectangles.h

```
#ifndef SEARCH_RECTANGLES_H
#define SEARCH_RECTANGLES_H
#include "png_structs.h"
```

```
void find_rectangles(struct Png* image, png_byte* rectangle_color,
png_byte* line_color, int line_width);
void search_rectangle(struct Png* image, Edge start, Rectangle*
rectangle, png_byte* rectangle_color);
int find_down(struct Png* image, Edge start, Edge* diagonal, png_byte*
rectangle_color);
int find_right(struct Png* image, Edge start, Edge* diagonal, png_byte*
rectangle_color);
Edge find_diagonal(struct Png* image, Edge start, png_byte*
rectangle_color);
int next_step_to_diagonal(struct Png* image, Edge start, Edge diagonal,
png_byte* rectangle_color);
int next_step_to_right(struct Png* image, Edge start, Edge diagonal,
png_byte* rectangle_color);
int next_step_to_down(struct Png* image, Edge start, Edge diagonal,
png_byte* rectangle_color);

#endif // SEARCH_RECTANGLES_H
```

Файл src/course_work/copy_frame.cpp

```
#include "png_structs.h"
#include "pixel_functions.h"

void copy_frame(struct Png* image, FrameCopy coords){
    int i, j;

    int frame_height = coords.right_bottom.y - coords.left_top.y;
    int frame_width = coords.right_bottom.x - coords.left_top.x;



    png_byte** frame = (png_byte**)malloc(sizeof(png_byte*) *
frame_height);
    for (i = 0; i < frame_height; i++){
        frame[i] = (png_byte*)malloc(sizeof(png_byte) * frame_height *
image->pixel_width);
    }
```

```
    for (i = coords.left_top.y; i < coords.left_top.y + frame_height;
i++){
        png_byte* row = image->row_pointers[i];
        png_byte* frame_row = frame[i - coords.left_top.y];
        for (j = coords.left_top.x; j < coords.left_top.x + frame_width;
j++){
            png_byte* ptr = &(row[j * image->pixel_width]);
            png_byte* frame_ptr = &(frame_row[(j - coords.left_top.x) *
image->pixel_width]);
            replace(frame_ptr, ptr, image->pixel_width);
        }
    }

    for (i = coords.destination.y; i - coords.destination.y <
frame_height; i++){
        if (i >= image->height || i < 0){
            continue;
        }
        png_byte* row = image->row_pointers[i];
        png_byte* frame_row = frame[i - coords.destination.y];
        for (j = coords.destination.x; j - coords.destination.x <
frame_width; j++){
            if (j >= image->width || j < 0){
                continue;
            }
            png_byte* ptr = &(row[j * image->pixel_width]);
            png_byte* frame_ptr = &(frame_row[(j - coords.destination.x)
* image->pixel_width]);
            replace(ptr, frame_ptr, image->pixel_width);
        }
    }

    for (i = 0; i < frame_height; i++){
        free(frame[i]);
    }
    free(frame);
}
```

Файл src/course_work/dialog.cpp

```
#include "dialog.h"
#include "ui_dialog.h"

Dialog::Dialog(int width, int height, QWidget *parent) :
    QDialog(parent),
    ui(new Ui::Dialog)
{
    ui->setupUi(this);
    ui->right_bottom_x->setValidator(new QIntValidator(0, width, this));
    ui->right_bottom_y->setValidator(new QIntValidator(0, height, this));
    ui->destination_x->setValidator(new QIntValidator(0, width, this));
    ui->destination_y->setValidator(new QIntValidator(0, height, this));
    ui->pushButton->setEnabled(false);

    ui->left_top_x->setValidator(new QIntValidator(0, width, this));
    ui->left_top_y->setValidator(new QIntValidator(0, height, this));

    connect(ui->left_top_x, SIGNAL(textChanged(QString)), this,
SLOT(onTextchanged(QString)));
    connect(ui->left_top_y, SIGNAL(textChanged(QString)), this,
SLOT(onTextchanged(QString)));
    connect(ui->right_bottom_x, SIGNAL(textChanged(QString)), this,
SLOT(onTextchanged(QString)));
    connect(ui->right_bottom_y, SIGNAL(textChanged(QString)), this,
SLOT(onTextchanged(QString)));
    connect(ui->destination_x, SIGNAL(textChanged(QString)), this,
SLOT(onTextchanged(QString)));
    connect(ui->destination_y, SIGNAL(textChanged(QString)), this,
SLOT(onTextchanged(QString)));
}

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

void Dialog::clean(QLineEdit* obj){
```

```
    QString cur = obj->text();
    if (cur.length() == 0){
        return;
    }
    if (cur.at(0) == '0'){
        obj->setText("0");
    }
}

int Dialog::check(){
   clean(ui->left_top_x);
   clean(ui->left_top_y);
   clean(ui->right_bottom_x);
   clean(ui->right_bottom_y);
   clean(ui->destination_x);
   clean(ui->destination_y);
   return !(ui->left_top_x->text().isEmpty() || ui->left_top_y-
>text().isEmpty() || \
            ui->right_bottom_x->text().isEmpty() || ui->right_bottom_y-
>text().isEmpty() || \
            ui->destination_x->text().isEmpty() || ui->destination_y-
>text().isEmpty());
}

void Dialog::onTextchanged(const QString &text){
    if (check()){
        ui->pushButton->setEnabled(true);
    }
}

void Dialog::on_pushButton_clicked()
{
    *(was_closed) = 0;
    frame_copy->left_top.x = ui->left_top_x->text().toInt();
    frame_copy->left_top.y = ui->left_top_y->text().toInt();
    frame_copy->right_bottom.x = ui->right_bottom_x->text().toInt();
    frame_copy->right_bottom.y = ui->right_bottom_y->text().toInt();
    frame_copy->destination.x = ui->destination_x->text().toInt();
```

```
    frame_copy->destination.y = ui->destination_y->text().toInt();
    close();
}
```

Файл src/course_work/get_ornament.cpp

```
#include "png_structs.h"
#include "pixel_functions.h"

void get_frame(struct Png* image, png_byte* line_color, int line_width,
Rectangle rectangle){
    // define region of frame and rectangle

    Rectangle region;
    region.left_top.x = rectangle.left_top.x - line_width;
    region.left_top.y = rectangle.left_top.y - line_width;

    region.right_bottom.x = rectangle.right_bottom.x + line_width;
    region.right_bottom.y = rectangle.right_bottom.y + line_width;



    int i, j;
    for (i = region.left_top.y; i <= region.right_bottom.y; i++){
        if (i >= image->height || i < 0){
            continue;
        }
        png_byte* row = image->row_pointers[i];
        for (j = region.left_top.x; j <= region.right_bottom.x; j++){
            if (j >= image->width || j < 0){
                continue;
            }
            png_byte* ptr = &(row[j * image->pixel_width]);

            if (!in(i, j, rectangle)){
                replace(ptr, line_color, image->pixel_width);
            }
        }
    }
}
```

```
void get_ornament(struct Png* image, int ornament_width, png_byte* color,
png_byte* second_color){
    Rectangle old_image;
    int i;

    old_image.left_top.x = ornament_width;
    old_image.left_top.y = ornament_width;

    old_image.right_bottom.x = image->width + ornament_width - 1;
    old_image.right_bottom.y = image->height + ornament_width - 1;

    int new_height = image->height + 2 * ornament_width;
    int new_width = image->width + 2 * ornament_width;
    png_byte** new_image = (png_byte**)malloc(sizeof(png_byte*) *
new_height * image->pixel_width);
    for (i = 0; i < new_height; i++){
        new_image[i] = (png_byte*)malloc(sizeof(png_byte) * new_width *
image->pixel_width);
    }


    for (i = 0; i < new_height; i++){
        png_byte* new_row = new_image[i];
        for (int j = 0; j < new_width; j++){
            png_byte* new_pixel = &(new_row[j * image->pixel_width]);

            if (in(i, j, old_image)){

                png_byte* old_row = image->row_pointers[i -
ornament_width];
                png_byte* old_pixel = &(old_row[(j - ornament_width) *
image->pixel_width]);
                replace(new_pixel, old_pixel, image->pixel_width);
            } else {
                replace(new_pixel, color, image->pixel_width);
            }
```

```
        }
    }

    for (i = 0; i < image->height; i++){
        free(image->row_pointers[i]);
    }
    free(image->row_pointers);

    image->row_pointers = new_image;
    image->width = new_width;
    image->height = new_height;



    Rectangle first_frame;
    first_frame.left_top.x = ornament_width / 3;
    first_frame.left_top.y = ornament_width / 3;

    first_frame.right_bottom.x = image->width - ornament_width / 3 - 1;
    first_frame.right_bottom.y = image-> height - ornament_width / 3 - 1;

    get_frame(image, second_color, ornament_width / 3, first_frame);
    get_frame(image, second_color, ornament_width / 3, old_image);
}
```

Файл src/course_work/imageviewer.cpp

```
#include "imageviewer.h"
#include "ui_imageviewer.h"
#include <QDebug>
#define NUMCHANNELS 3

ImageViewer::ImageViewer(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::ImageViewer)
{
    ui->setupUi(this);
    connect(ui->single_color, SIGNAL(triggered()), this,
SLOT(set_ornament()));
```

```
    connect(ui->double_color, SIGNAL(triggered()), this,
SLOT(set_ornament()));
}

ImageViewer::~ImageViewer()
{
    delete ui;
    delete png_file;
    delete scene;
}

int ImageViewer::valid(struct FrameCopy* fcopy){
    if (fcopy->left_top.x >= fcopy->right_bottom.x || fcopy->left_top.y
>= fcopy->right_bottom.y){
        QMessageBox::critical(this, "Ошибка", "Левый верхний угол должен
находится выше правого нижнего угла.");
        return 0;
    }
    return 1;
}

void ImageViewer::refresh(){
    write_png_file(png_file, filename);
    start();
}


void ImageViewer::start(){

    if (read_png_file(png_file, filename, this)){
        image.load(filename);
        int width = image.width();
        int height = image.height();
        ui->imgView->resize(width, height);
        ImageViewer::resize(width, height);
        scene->addPixmap(image);
        scene->setSceneRect(image.rect());
        ui->imgView->setScene(scene);
```

```
    } else {
        close();
    }
}

int ImageViewer::get_color(png_byte color[], QString info){
    QColor setting_color = QColorDialog::getColor(Qt::white, this, info);
    if (!setting_color.isValid()){
        return 0;
    }
    color[0] = setting_color.red();
    color[1] = setting_color.green();
    color[2] = setting_color.blue();
    return 1;
}

void ImageViewer::set_ornament()
{
    png_byte first_ornament_color[NUMCHANNELS];
    if (!get_color(first_ornament_color, "Выберите цвет")){
        return;
    }

    png_byte second_ornament_color[NUMCHANNELS];

    QAction* s = (QAction*)sender();
    if (s->text() == "два цвета"){
        if (!get_color(second_ornament_color, "Выберите второй цвет")){
            return;
        }
    } else {
        for (int i = 0; i < 2; i++){
            second_ornament_color[i] = first_ornament_color[i];
        }
    }

    int ornament_width = QInputDialog::getInt(this, "Выберите ширину
рамки", "ширина", 0, 1);
```

```
    get_ornament(png_file, ornament_width, first_ornament_color,
second_ornament_color);


    refresh();
}


void ImageViewer::on_actionsearch_rectangles_triggered()
{
    png_byte rec_color[NUMCHANNELS];
    if (!get_color(rec_color, "Выберите цвет прямоугольников")){
        return;
    }

    png_byte frame_color[NUMCHANNELS];
    if (!get_color(frame_color, "Выберите цвет обводки")){
        return;
    }

    int frame_width = QInputDialog::getInt(this, "Выберите ширину рамки",
"ширина", 0, 1);
    find_rectangles(png_file, rec_color, frame_color, frame_width);

    refresh();
}


void ImageViewer::on_actionchange_color_triggered()
{
    png_byte replaced_color[NUMCHANNELS];
    png_byte color_for_replace[NUMCHANNELS];
    if (!get_color(replaced_color, "Выберите цвет, который надо
заменить.")){
        return;
    }
```

```
    if (!get_color(color_for_replace, "Выберите цвет, на который будет
заменен предыдущий.")){
        return;
    }
    change_color(png_file, replaced_color, color_for_replace);
    refresh();
}


void ImageViewer::on_actioncopy_frame_triggered()
{
    FrameCopy* fcopy = new FrameCopy;
    Dialog* new_window = new Dialog(png_file->width, png_file->height);
    int* was_closed = new int;
    new_window->setAttribute(Qt::WA_DeleteOnClose);
    new_window->set_frame(fcopy, was_closed);
    new_window->setModal(true);
    new_window->exec();
    if (!(*(was_closed))){
        // check valid
        if (valid(fcopy)){
            copy_frame(png_file, *fcopy);
            refresh();
        }
    }
    delete was_closed;
    delete fcopy;
}
```

Файл src/course_work/main.cpp

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
```

```
    MainWindow w;
    w.show();
    return a.exec();
}
```

Файл src/course_work/mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "imageviewer.h"
#include <QDebug>

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

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



void MainWindow::on_pushButton_clicked()
{
    QString filepath = QFileDialog::getOpenFileName(this,
                       tr("Выберите изображения"), "", tr("Image Files
(*.png)"));

    if (filepath.length()){
        ImageViewer* imgV = new ImageViewer;
        imgV->show();
        imgV->setAttribute(Qt::WA_DeleteOnClose);
        imgV->init();
        imgV->setFile(filepath);
        imgV->start();
```

```
    }
}


void MainWindow::on_pushButton_2_clicked()
{
    QString info = "Вас приветствует программа обработки изображений!\n \
    Данная программа позволяет обрабатывать файлы типа PNG с форматом
пикселей RGB.\n \
    Программа имеет следующие функции:\n \
    >> Скопировать заданную область \n \
    >> Заменить все пиксели одного цвета на другой \n \
    >> Сделать рамку\n \
    >> Найти все прямоугольники заданного цвета";

    QMessageBox::information(this, "Справка", info);
}
```

Файл src/course_work/pixel_functions.cpp

```
#include "png_structs.h"

int equal(png_byte* image_pixel, png_byte* color, int width){
    /* compare two pixels */
    for (int i = 0; i < width; i++){
        if (image_pixel[i] != color[i]){
            return 0;
        }
    }
    return 1;
}

void replace(png_byte* image_pixel, png_byte* color, int width){
    /* replaces one pixel with another */
    for (int i = 0; i < width; i++){
        image_pixel[i] = color[i];
    }
}
```

```
void change_color(struct Png* image, png_byte* replaced_color, png_byte*
color_to_replace){
    int x, y;
    for (y = 0; y < image->height; y++){
        png_byte* row = image->row_pointers[y];
        for (x = 0; x < image->width; x++){
            png_byte* ptr = &(row[x * image->pixel_width]);

            if (equal(ptr, replaced_color, image->pixel_width)){
                replace(ptr, color_to_replace, image->pixel_width);
            }
        }
    }
}

int in(int y, int x, Rectangle rectangle){
    return (rectangle.left_top.y <= y && y <= rectangle.right_bottom.y)
&& \
    (rectangle.left_top.x <= x && x <= rectangle.right_bottom.x);
}

int checked(int y, int x, Rectangle* rectangles, int len){
    for (int i = 0; i < len; i++){
        if (in(y, x, rectangles[i])){
            return 1;
        }
    }

    return 0;
}
```

Файл src/course_work/read_write_functions.cpp

```
#include "png_structs.h"
#include <QMessageBox>
#include <QMainWindow>

int read_png_file(struct Png* image, QString file_name, QMainWindow* win)
{
```

```
    int y;
    png_byte header[8];    // 8 is the maximum size that can be checked
    QByteArray ba = file_name.toLocal8Bit();
    const char *c_str = ba.data();

    /* open file and test for it being a png */
    FILE *fp = fopen(c_str, "rb");
    if (!fp){
        QMessageBox::critical(win, "Ошибка", "Файл не может быть
открыт.");
        return 0;
    }

    fread(header, 1, 8, fp);
    if (png_sig_cmp(header, 0, 8)){
        QMessageBox::critical(win, "Ошибка", "Тип файла не PNG");
        return 0;
    }

    /* initialize stuff */
    image->png_ptr = png_create_read_struct(PNG_LIBPNG_VER_STRING, NULL,
NULL, NULL);

    if (!image->png_ptr){
        QMessageBox::critical(win, "Ошибка", "Ошибка чтения файла.
Проверьте корректность переданного файла.");
        return 0;
    }

    image->info_ptr = png_create_info_struct(image->png_ptr);
    if (!image->info_ptr){
        QMessageBox::critical(win, "Ошибка", "Ошибка чтения файла.
Проверьте корректность переданного файла.");
        return 0;
    }

    if (setjmp(png_jmpbuf(image->png_ptr))){
```

```
        QMessageBox::critical(win, "Ошибка", "Ошибка чтения файла.
Проверьте корректность переданного файла.");
        return 0;
    }

    png_init_io(image->png_ptr, fp); // init input/output
    png_set_sig_bytes(image->png_ptr, 8); // libpng needs to know that
some chuncks were readen (in png_sig_cmp)

    png_read_info(image->png_ptr, image->info_ptr); // process chuncks

    /* now we can easy query the info structure above
    this do the following functions: */

    image->width = png_get_image_width(image->png_ptr, image->info_ptr);
// width
    image->height = png_get_image_height(image->png_ptr, image-
>info_ptr); // height
    image->color_type = png_get_color_type(image->png_ptr, image-
>info_ptr); // color type
    image->bit_depth = png_get_bit_depth(image->png_ptr, image-
>info_ptr); // bit depth



    png_read_update_info(image->png_ptr, image->info_ptr); // after that
we can read image row by row

    /* read file */
    if (setjmp(png_jmpbuf(image->png_ptr))){
        QMessageBox::critical(win, "Ошибка", "Ошибка чтения файла.
Проверьте корректность переданного файла.");
        return 0;
    }

    // allocate memory for storing image
    image->row_pointers = (png_bytep *) malloc(sizeof(png_bytep) * image-
>height);
    for (y = 0; y < image->height; y++)
```

```
        image->row_pointers[y] = (png_byte *)
malloc(png_get_rowbytes(image->png_ptr, image->info_ptr));

    // reading image
    png_read_image(image->png_ptr, image->row_pointers);

    if (png_get_color_type(image->png_ptr, image->info_ptr) ==
PNG_COLOR_TYPE_RGB){
        image->pixel_width = 3;
    } else {
        QMessageBox::critical(win, "Ошибка", "Входной файл должен
принадлежать формату RGBA.");
        return 0;
    }
    fclose(fp);
    return 1;
}

void write_png_file(struct Png* image, QString file_name) {
    int y;

    QByteArray ba = file_name.toLocal8Bit();
    const char *c_str = ba.data();

    /* create file */
    FILE *fp = fopen(c_str, "wb");


    /* initialize stuff */
    image->png_ptr = png_create_write_struct(PNG_LIBPNG_VER_STRING, NULL,
NULL, NULL);

    image->info_ptr = png_create_info_struct(image->png_ptr);

    png_init_io(image->png_ptr, fp);


    /* now we need fill in the png_info structure with all data
```

```
    we wish to write before the actual image. */
    png_set_IHDR(image->png_ptr, image->info_ptr, image->width, image-
>height,
                 image->bit_depth, image->color_type, PNG_INTERLACE_NONE,
                 PNG_COMPRESSION_TYPE_BASE, PNG_FILTER_TYPE_BASE);

    png_write_info(image->png_ptr, image->info_ptr);


    png_write_image(image->png_ptr, image->row_pointers);

    png_write_end(image->png_ptr, NULL);

    /* cleanup heap allocation */
    for (y = 0; y < image->height; y++)
        free(image->row_pointers[y]);
    free(image->row_pointers);

    fclose(fp);
}
```

Файл src/course_work/search_rectangles.cpp

```
#include "png_structs.h"
#include "pixel_functions.h"
#include "get_ornament.h"

int next_step_to_down(struct Png* image, Edge start, Edge diagonal,
png_byte* rectangle_color){
    // check row
    if (diagonal.y + 1 >= image->height){
        return 0;
    }
    png_byte* row = image->row_pointers[diagonal.y + 1];
    for (int i = start.x; i <= diagonal.x; i++){
        if (!equal(rectangle_color, &(row[i * image->pixel_width]),
image->pixel_width)){
            return 0;
        }
```

```
    }
    return 1;
}

int next_step_to_right(struct Png* image, Edge start, Edge diagonal,
png_byte* rectangle_color){
    // check column
    if (diagonal.x + 1 >= image->width){
        return 0;
    }
    for (int i = start.y; i <= diagonal.y; i++){
        png_byte* row = image->row_pointers[i];
        if (!equal(rectangle_color, &(row[(diagonal.x + 1) * image-
>pixel_width]), image->pixel_width)){
            return 0;
        }
    }
    return 1;
}

int next_step_to_diagonal(struct Png* image, Edge start, Edge diagonal,
png_byte* rectangle_color){

    png_byte* row = image->row_pointers[diagonal.y + 1];
    if (!equal(rectangle_color, &(row[(diagonal.x + 1) * image-
>pixel_width]), image->pixel_width)){
        return 0;
    }

    if (!next_step_to_right(image, start, diagonal, rectangle_color)){
        return 0;
    }

    if (!next_step_to_down(image, start, diagonal, rectangle_color)){
        return 0;
    }

    return 1;
```

```
}

Edge find_diagonal(struct Png* image, Edge start, png_byte*
rectangle_color){
    Edge diagonal;
    diagonal.x = start.x;
    diagonal.y = start.y;
    while (next_step_to_diagonal(image, start, diagonal,
rectangle_color)){
        diagonal.x++;
        diagonal.y++;
        if ((diagonal.x + 1) >= image->width || (diagonal.y + 1) >=
image->height){
            break;
        }
    }
    return diagonal;
}

int find_right(struct Png* image, Edge start, Edge* diagonal, png_byte*
rectangle_color){
    int i = 0;
    while (next_step_to_right(image, start, *diagonal, rectangle_color)){
        i = 1;
        diagonal->x += 1;
    }
    return i;
}

int find_down(struct Png* image, Edge start, Edge* diagonal, png_byte*
rectangle_color){
    int i = 0;
    while (next_step_to_down(image, start, *diagonal, rectangle_color)){
        i = 1;
        diagonal->y += 1;
    }
    return i;
}
```

```
void search_rectangle(struct Png* image, Edge start, Rectangle*
rectangle, png_byte* rectangle_color){
    // check coordinates of diagonal
    Edge diagonal = find_diagonal(image, start, rectangle_color);


    if (!find_right(image, start, &diagonal, rectangle_color)){
        find_down(image, start, &diagonal, rectangle_color);
    }

    rectangle->left_top.x = start.x;
    rectangle->left_top.y = start.y;

    rectangle->right_bottom.x = diagonal.x;
    rectangle->right_bottom.y = diagonal.y;
}

void find_rectangles(struct Png* image, png_byte* rectangle_color,
png_byte* line_color, int line_width){
    Edge start;
    int num_rectangles = 10;
    int current_rectangle = 0;
    int i, j;
    Rectangle* rectangles = (Rectangle*)malloc(sizeof(Rectangle) *
num_rectangles);
    for (i = 0; i < image->height; i++){
        png_byte* row = image->row_pointers[i];
        for (j = 0; j < image->width; j++){
            png_byte *ptr = &(row[j * image->pixel_width]);

            if (equal(ptr, rectangle_color, image->pixel_width)){
                if (!checked(i, j, rectangles, current_rectangle)){
                    // creating new rectangle
                    // check size of array
                    if (current_rectangle == num_rectangles){
                        num_rectangles *= 2;
```

```
                        rectangles = (Rectangle*)realloc(rectangles,
sizeof(Rectangle) * num_rectangles);
                    }
                    start.y = i;
                    start.x = j;
                    search_rectangle(image, start,
&rectangles[current_rectangle++], rectangle_color);

                }
            }
        }
    }


    // set frames of rectangles
    for (i = 0; i < current_rectangle; i++){
        get_frame(image, line_color, line_width, rectangles[i]);
    }

    free(rectangles);

}
```