

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

**КУРСОВАЯ РАБОТА**  
**по дисциплине «Программирование»**  
**ТЕМА: РАБОТА С ИЗОБРАЖЕНИЯМИ**

Студент гр. 3342

Легалов В. В.

Преподаватель

Глазунов С. А.

Санкт-Петербург

2024

## **ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ**

Студент: Легалов В. В.

Группа: 3342

Тема работы: Работа с изображениями

Вариант 5.12

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

Программа должна реализовывать весь следующий функционал по обработке png-файла

Общие сведения

Формат картинки PNG (рекомендуем использовать библиотеку libpng) без сжатия

файл может не соответствовать формату PNG, т.е. необходимо проверка на PNG формат. Если файл не соответствует формату PNG, то программа должна завершиться с соответствующей ошибкой.

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

все поля стандартных PNG заголовков в выходном файле должны иметь те же значения что и во входном (разумеется кроме тех, которые должны быть изменены).

Программа должна иметь следующую функции по обработке изображений:

Рисование квадрата. Флаг для выполнения данной операции: `--square``.  
Квадрат определяется:

Координатами левого верхнего угла. Флаг `--left_up``, значение задаётся в формате `left.up``, где `left` – координата по `x`, `up` – координата по `y`

Размером стороны. Флаг `--side_size`` На вход принимает число больше 0

Толщиной линий. Флаг `--thickness``. На вход принимает число больше 0

Цветом линий. Флаг `--color`` (цвет задаётся строкой `rrr.ggg.bbb``, где `rrr/ggg/bbb` – числа, задающие цветовую компоненту. пример `--color 255.0.0`` задаёт красный цвет)

Может быть залит или нет. Флаг `--fill``. Работает как бинарное значение: флага нет – `false` , флаг есть – `true`.

Цветом которым он залит, если пользователем выбран залитый. Флаг `--fill_color`` (работает аналогично флагу `--color``)

Поменять местами 4 куса области. Флаг для выполнения данной операции: `--exchange``. Выбранная пользователем прямоугольная область делится на 4 части и эти части меняются местами. Функционал определяется:

Координатами левого верхнего угла области. Флаг `--left_up``, значение задаётся в формате `left.up``, где `left` – координата по `x`, `up` – координата по `y`

Координатами правого нижнего угла области. Флаг `--right_down``, значение задаётся в формате `right.down``, где `right` – координата по `x`, `down` – координата по `y`

Способом обмена частей: “по кругу”, по диагонали. Флаг `--exchange_type``, возможные значения: `clockwise``, `counterclockwise``, `diagonals``

Находит самый часто встречаемый цвет и заменяет его на другой заданный цвет. Флаг для выполнения данной операции: `--freq_color``.

Функционал определяется:

Цветом, в который надо перекрасить самый часто встречаемый цвет.

Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где `rrr/ggg/bbb` – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)

Инверсия цвета в заданной области. Флаг для выполнения данной операции: `--inverse`. Функционал определяется

Координатами левого верхнего угла области. Флаг `--left_up`, значение задаётся в формате `left.up`, где `left` – координата по `x`, `up` – координата по `y`

Координатами правого нижнего угла области. Флаг `--right_down`, значение задаётся в формате `right.down`, где `right` – координата по `x`, `down` – координата по `y`

Каждую подзадачу следует вынести в отдельную функцию, функции сгруппировать в несколько файлов (например, функции обработки текста в один, функции ввода/вывода в другой). Сборка должна осуществляться при помощи `make` и `Makefile` или другой системы сборки

Разделы пояснительный записки: «Содержание», «Введение», «Классы», «Функции», «Заключение», «Список использованных источников», «Приложение А. Примеры работы программы», «Приложение Б. Исходный код программы».

Предполагаемый объем пояснительной записки:

Не менее 20 страниц.

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 20.05.2024

Дата защиты реферата: 22.05.2024

Студент

Легалов В. В.

Преподаватель

Глазунов С. А.

## **АННОТАЦИЯ**

Курсовая работа заключается в создании программы для обработки изображений в формате PNG на языке C++. Она представляет собой набор компонентов для обработки файлов формата PNG, включающих функции рисования квадрата, перестановки фрагментов изображения в области, замены самого часто встречающегося цвета другим и инверсии цвета в заданной области. Структура проекта включает каталог src для файлов исходного кода, заголовочных файлов, Makefile для сборки проекта и файл help со справкой к программе. Для сборки проекта используется команда make.

Пример работы программы приведен в приложении А.

Исходный код программы приведен в приложении Б.

## **SUMMARY**

The course work consists of creating a program for processing images in PNG format in C++. It is a set of components for processing PNG files, including functions for drawing a square, rearranging image fragments in an area, replacing the most frequently occurring color with another, and inverting a color in a given area. The project structure includes a src directory for source code files, header files, a Makefile for building the project, and a help file with help for the program. To build the project, use the make command.

An example of how the program works is given in appendix A.

The source code of the program is given in appendix B.

## СОДЕРЖАНИЕ

Введение.....	7
1. Классы.....	8
1.1 Класс dataPNG.....	8
1.2 Класс Chunck.....	9
1.3 Структура rgb.....	9
1.4 Класс Image.....	9
1.5 Класс arguments.....	10
2. Функции.....	11
2.1 Функция main.....	11
2.2 Функция getarguments.....	11
2.3 Функция drawSquare.....	11
2.3 Функция exchange.....	12
2.4 Функция freqcolor.....	12
2.5 Функция inverserect.....	12
Заключение.....	13
Список использованных источников.....	14
Приложение А.....	15
Приложение Б.....	17

## **ВВЕДЕНИЕ**

Цель работы: написать программу на языке C++, которая считывает изображение и обрабатывает его требуемым пользователем образом. Для этого требуется реализовать:

- Загрузку, хранение изображений из файла и запись изображения в файл;
- Считывание аргументов из командной строки и их валидация;
- Функции для обработки изображения требуемым образом

## 1. КЛАССЫ

Для решения поставленных задач были созданы следующие классы: dataPNG, Chunck, rgb, Image.

### 1.1 Класс dataPNG

Класс dataPng представляет изображение в формате PNG в том виде, в котором изображение храниться на устройстве.

Класс содержит следующие поля:

width — количество пикселей по ширине изображения

height — количество пикселей по высоте изображения

depth — битовая глубина изображения, количество бит на один цветовой канал

colortype — тип цвета изображения

pixelsize — количество байт, хранящих значение цвета одного пикселя

chunks — массив чанков, хранящих все сведения об изображении

В данном классе реализованы публичные методы для считывания и записи изображения:

readPNG(FILE \*) - считывает изображение из файла и заполняет поля класса

decodeImage() - переводит хранящиеся в чанках данные изображения в пригодный для обработки массив пикселей

updateData(Image &) - обновляет содержимое чанков изображения на основе поданного массива пикселей

writePNG(FILE \*) - записывает данные изображения в файл

printdata() - выводит информацию об изображении

Также, подзадачи при считывании и записи изображения выведены в приватные методы:

uncompressdata(uint8\_t\*&, uint64\_t&) - возвращает массив байт разжатых данных о пикселях изображения



`compressdata(uint8_t*&, uint64_t&)` - возвращает массив байт сжатых данных о пикселях изображения

`fillimage(uint8_t*, uint64_t)` — составляет из строки байт двумерный массив и разфильтровывает содержимое, возвращает полученный массив пикселей

`recon(uint8_t **, uint8_t, uint64_t, uint64_t)` — приводит содержимое зафильтрованного байта к исходному значению

## 1.2 Класс **Chunk**

Класс `Chunk`, вложенный в `dataPng`, хранит некоторую информацию об изображении. Включает в себя следующие поля:

`name` — число, хранящее информацию о типе чанка

`crc` — число, хранящее контрольную сумму для проверки сохранности содержимого

`data` — указатель на массив байт, в котором хранятся данные чанка

`length` — длина массива данных чанка

В данном классе реализованы следующие методы:

`setFromFile(FILE *)` - считывает из файла значения чанка

`writeToFile(FILE *)` - записывает в файл содержимое всех полей чанка

`calcCRC()` - вычисляет значение контрольной суммы для данных чанка

## 1.3 Структура **rgb**

Содержит информацию о цвете в трёх полях:

`r` — значение цвета по красному каналу

`g` — значение цвета по зелёному каналу

`b` — значение цвета по синему каналу

## 1.4 Класс **Image**

Содержит информацию об изображении в виде матрицы пикселей. Информация хранится в полях:

`height` — высота изображения

width — ширина изображения

pixelsize — количество байт, используемых для хранения одного пикселя

bitmap — матрица байт цветов пикселей

Для работы с содержимым изображения применяются методы:

setpixel(int, int, rgb &) — устанавливает заданный цвет пикселя по заданным координатам

getpixel(int, int) — возвращает указатель на пиксель в заданных координатах

checkCoordinats(int, int) — проверяет существование пикселя в заданных координатах

### **1.5 Класс arguments**

Содержит информацию о введенных пользователем параметрах:

ishelp — запрос справки о программе

isinfo — запрос информации об изображении

isfill — требование закрасить фигуру

number — номер операции, которую требуется выполнить

x1 — координата вдоль оси X левого верхнего угла области

y1 — координата вдоль оси Y левого верхнего угла области

x2 — координата вдоль оси X правого нижнего угла области

y2 — координата вдоль оси Y правого нижнего угла области

length — длина стороны фигуры

thickness — толщина линии

type\_exchange — тип обмена частей

color[3] — три числа типа int хранящие информацию о цвете

fill\_color[3] — три числа типа int хранящие информацию о цвете

input\_path — строка пути к файлу с входными значениями

output\_path — строка пути к файлу с выходными значениями

## 2. ФУНКЦИИ

### 2.1 Функция `main`

Функция `main` является главной функцией программы, в ней выполняются следующие шаги:

Инициализация объекта класса `arguments`, в который с помощью функции `getarguments` записываются считанные данные.

Если пользователем была введена соответствующая опция, выводится справка о программе.

Инициализация объекта класса `dataPNG`, считывание данных изображения из файла.

Если пользователем была введена соответствующая опция, выводится информация об изображении.

Инициализация объекта класса `Image`, заполнение матрицы пикселей из считанных данных изображения.

Выполнение заданного преобразования над изображением.

Запись изображения в файл.

### 2.2 Функция `getarguments`

Принимает значения `argc` и `argv`, которые были получены функцией `main`, инициализирует объект класса `arguments` и при помощи функции `getopt_long` в цикле определяет значение введенных пользователем флагов и их аргументы. Возвращает указатель на заполненный объект класса `arguments`. В случае, если переданный параметр или флаг является недопустимым, завершает работу программы с ошибкой.

### 2.3 Функция `drawSquare`

Рисует квадрат на изображении по заданным параметрам расположения левого верхнего угла, толщины линии, длины стороны, цвета контура и заливки. Функция имеет перегрузку, в случае, если заливка квадрата не

требуется, аргумент заливки квадрата не указывается, тогда рисуется только контур.

### **2.3 Функция `exchange`**

Меняет местами четыре фрагмента изображения из заданной области. Обмен частей осуществляется одним из трех возможных способов: по кругу, по часовой стрелке или против, или по диагонали. Для выполнения этой операции создаётся буффер, в который копируется заданная область при помощи функции `sorugest`, далее каждый фрагмент поочерёдно вставляется в изображение.

Функция `sorugest` принимает указатели на массив, в который нужно скопировать данные и массив из которого берутся значения, а также координаты начала копирования для обоих массивов и размеры копируемой области.

### **2.4 Функция `freqcolor`**

Обходит всё изображение, записывая количество каждого встреченного цвета в бинарное дерево `map`, используя в качестве ключа значение цвета пикселя. Из полученного дерева берётся цвет, который встречался чаще всего и в изображении все пиксели этого цвета перекрашиваются на указанный.

### **2.5 Функция `inverserect`**

Принимает координаты левого верхнего и правого нижнего углов области, в которой требуется инвертировать цвета и при помощи функции `inversepixel` перекрашивает каждый пиксель в заданном прямоугольнике.

Функция `inversepixel` побитово инвертирует значение каждого поля указанного пикселя изображения.

Разработанный программный код см. в приложении Б.

## **ЗАКЛЮЧЕНИЕ**

Была написана программа на языке C++, которая считывает изображение, изменяет его содержимое требуемым пользователем образом и записывает результат в файл. Для этого были реализованы:

Загрузка, хранение изображений из файла и запись изображения в файл;

Ввод аргументов из командной строки;

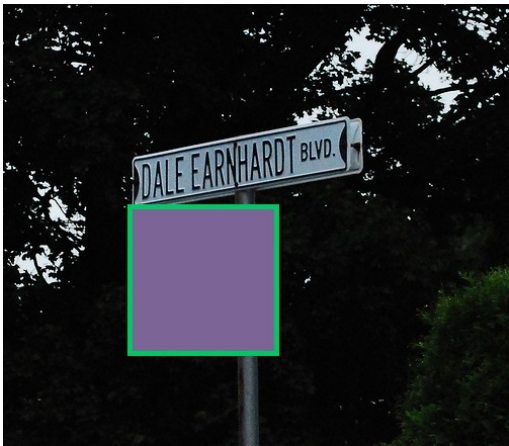
Ввод аргументов из командной строки был осуществлен с помощью функции `getopt_long` из стандартной библиотеки Си. Для распаковки и упаковки данных изображения при чтении и записи были использованы функции из `zlib`.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

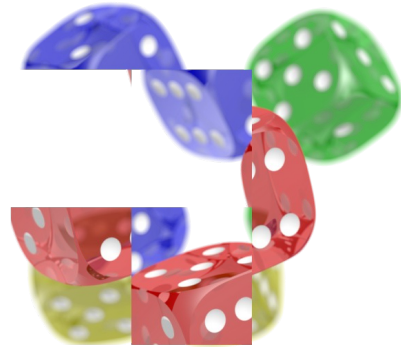
1. Статья по функционалу C++. URL: <https://metanit.com/cpp/tutorial/2.14.php>.
2. Мануал по использованию getopt. URL:  
[https://www.gnu.org/software/libc/manual/html\\_node/Getopt.html](https://www.gnu.org/software/libc/manual/html_node/Getopt.html).
3. Portable Network Graphics (PNG) Specification (Third Edition) URL:  
<https://www.w3.org/TR/png-3>

# **ПРИЛОЖЕНИЕ А** **ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ**

Изображение test1.png	Изображение test2.png
	

Входные данные	Выходные данные
<pre>./cw --square --left_up 125.200 --thickness 5 --side_size 150 --color 0.200.100 --fill --fill_color 125.100.150 test2.png</pre>	

```
./cw --exchange --left_up  
100.120 --right_down 450.520 --  
exchange_type clockwise test1.png
```



```
./cw --inverse --left_up 280.210  
--right_down 520.380 test1.png
```



```
./cw --freq_color --color  
0.255.0 test2.png
```





## ПРИЛОЖЕНИЕ Б

### ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.cpp:

```
#include<iostream>
#include<cstring>
#include<utility>
#include<math.h>

#include "image.hpp"
#include "input_arguments.hpp"
#include "pngdata.hpp"
#include "square.hpp"
#include "inverse.hpp"
#include "exchange.hpp"
#include "freqcolor.hpp"

void rhombus(Image *image, rgb color){
    int height = image->height;
    int width = image->width;

    int x_center = width / 2;
    int y_center = height / 2;
    float ta = (float)height / (float)width ;

    for(int i = 0; i <= y_center; ++i)
        for(int j = 0; j <= x_center; ++j){
            if(i >= int(ta * j)){
                image->setpixel(j + x_center, i, color);
                image->setpixel(x_center - j, i, color);
            }
            if(i <= int(ta * j)){
                image->setpixel(j, i + y_center, color);
                image->setpixel(width - j, i + y_center, color);
            }
        }
    }

int main(int argc, char** argv){
    arguments *args = getarguments(argc, argv);

    FILE *file;
    if(args->ishelp){
        file = fopen("help", "rb");
        if (!file) {
            std::cout << "File \"help\" is not found.\n";
            exit(49);
        }
    }
```

```

        char ch = (char) fgetc(file);
        while (ch != EOF) {
            printf("%c", ch);
            ch = (char) fgetc(file);
        }
        fclose(file);
        exit(0);
    }
    file = fopen(args->input_path, "rb");
    if(!file){
        std::cout << "Input file is not found\n";
        exit(49);
    }
    dataPNG source;
    source.readPNG(file);
    fclose(file);
    if(args->isinfo){
        source.printdata();
        exit(0);
    }

    Image *image = source.decodeImage();

    switch(args->number){
        case 1: {
            rgb color, fill_color;
            if(args->color[0] == -1){
                std::cout << "Color is not defined\n";
                exit(42);
            }
            color = rgb(args->color[0], args->color[1], args->
>color[2]);

            if(args->thickness <= 0){
                std::cout << "Invalid thickness value\n";
                exit(42);
            }
            if(args->length <= 0){
                std::cout << "Invalid side size value\n";
                exit(42);
            }
            if(args->isfill){
                if(args->fill_color[0] == -1){
                    std::cout << "Fill color is not defined\n";
                    exit(42);
                }
                fill_color = rgb(args->fill_color[0], args->
>fill_color[1], args->fill_color[2]);
                drawSquare(image, args->x1, args->y1, args->length,
args->thickness, color, fill_color);
            }
            else
                drawSquare(image, args->x1, args->y1, args->length,
args->thickness, color);
            break;

```

```

    }
    case 2:
        if (args->x1 > args->x2)
            std::swap(args->x1, args->x2);
        if (args->y1 > args->y2)
            std::swap(args->y1, args->y2);
        if(args->type_exchange){
            exchange(image, args->x1, args->y1, args->x2, args-
>y2, args->type_exchange);
        } else{
            std::cout << "Invalid exchange type\n";
            exit(42);
        }
        break;
    case 3: {
        rgb color;
        if(args->color[0] == -1){
            std::cout << "Color is not defined\n";
            exit(42);
        }
        color = rgb(args->color[0], args->color[1], args-
>color[2]);
        freqcolor(image, color);
        break;
    }
    case 4:
        if (args->x1 > args->x2)
            std::swap(args->x1, args->x2);
        if (args->y1 > args->y2)
            std::swap(args->y1, args->y2);
        inverserect(image, args->x1, args->y1, args->x2, args-
>y2);
        break;
    case 5:{
        rgb color;
        if(args->color[0] == -1){
            std::cout << "Color is not defined\n";
            exit(42);
        }
        color = rgb(args->color[0], args->color[1], args-
>color[2]);
        rhombus(image, color);
        break;
    }
    default:
        std::cout << "Undefined operation\n";
        exit(43);
}

source.updateData(*image);
delete image;
file = fopen(args->output_path, "wb");
source.writePNG(file);
fclose(file);
delete args;
return 0;

```

```
}
```

### Файл Makefile:

```
CC = g++

all: main.o pngdata.o image.o square.o inverse.o exchange.o
freqcolor.o input_arguments.o
    $(CC) main.o pngdata.o image.o input_arguments.o square.o
inverse.o exchange.o freqcolor.o -lz -o cw

main.o: main.cpp image.hpp
    $(CC) -c main.cpp

pngdata.o: pngdata.cpp pngdata.hpp
    $(CC) -c pngdata.cpp

image.o: image.cpp image.hpp
    $(CC) -c image.cpp

input_arguments.o: input_arguments.cpp input_arguments.hpp
    $(CC) -c input_arguments.cpp

square.o: square.cpp square.hpp
    $(CC) -c square.cpp

inverse.o: inverse.cpp inverse.hpp
    $(CC) -c inverse.cpp

exchange.o: exchange.cpp exchange.hpp
    $(CC) -c exchange.cpp

freqcolor.o: freqcolor.cpp freqcolor.hpp
    $(CC) -c freqcolor.cpp

clear:
    rm *.o
```

### Файл pngdata.hpp

```
#pragma once
#include<iostream>
#include<vector>
#include<cstring>
#include<zlib.h>
#include "image.hpp"

const uint8_t SIGNATUREPNG[8] = {0x89, 0x50, 0x4E, 0x47 , 0x0D,
0x0A, 0x1A, 0x0A};
#define IHDR 1380206665
#define IEND 1145980233
#define IDAT 1413563465
#define PLTE 1163152464
#define IDATSIZE 8192
```

```

uint32_t reversebyte(uint32_t x);
void checkpointer(void*);

class dataPNG{
public:
    class Chunk{
    public:
        uint32_t length;
        uint32_t name;
        uint32_t crc;
        uint8_t *data;
        Chunk();
        Chunk(uint32_t, uint32_t, uint8_t*);
        int setFromFile(FILE *);
        void writeInFile(FILE *);
        uint32_t calcCRC();
    };
    std::vector<Chunk> chunks;
    uint32_t width, height;
    uint8_t depth, colortype;
    uint8_t interlace, pixelsize;
    uint8_t *palette;
    uint32_t palettesize;
    int readPNG(FILE *);
    Image* decodImage();
    int updateData(Image &);
    int writePNG(FILE *);
    void printdata();
    ~dataPNG();
private:
    uint64_t sizedata();
    int uncompressdata(uint8_t*&, uint64_t&);
    int compressdata(uint8_t*&, uint64_t&);
    uint8_t** fillimage(uint8_t*, uint64_t);
    void recon(uint8_t **, uint8_t, uint64_t, uint64_t);
    uint8_t paethPredictor(int, int, int);
    void fillFromPalette(uint8_t **&);
};

```

Файл pngdata.cpp

```

#include "pngdata.hpp"

uint32_t reversebyte(uint32_t x){
    x = (x & 0x00FF00FF) << 8 | (x & 0xFF00FF00) >> 8;
    x = (x & 0x0000FFFF) << 16 | (x & 0xFFFF0000) >> 16;
    return x;
}

void checkpointer(void *ptr){
    if (ptr == NULL){
        std::cout << "Error: not enough memory\n";
        exit(48);
    }
}

```

```

}

dataPNG::Chunk::Chunk(){
    length = 0;
    name = 0;
    crc = 0;
    data = NULL;
}

dataPNG::Chunk::Chunk(uint32_t len, uint32_t str, uint8_t* data){
    length = len;
    name = str;
    if(length){
        this->data = new uint8_t[length];
        checkpointer(this->data);
        memcpy(this->data, data, length);
    }
    crc = calcCRC();
}

int
dataPNG::Chunk::setFromFile(FILE *file){
    fread(&length, 1, 4, file);
    length = reversebyte(length);
    fread(&name, 1, 4, file);
    if(length){
        if (data != NULL)
            delete[] data;
        data = new uint8_t[length];
        checkpointer(data);
        fread(data, 1, length, file);
    }
    fread(&crc, 1, 4, file);
    crc = reversebyte(crc);
    return 0;
}

void
dataPNG::Chunk::writeInFile(FILE *file){
    uint32_t buffer;
    buffer = reversebyte(length);
    fwrite(&buffer, 1, 4, file);
    fwrite(&name, 1, 4, file);
    if(length){
        fwrite(data, 1, length, file);
    }
    buffer = reversebyte(crc);
    fwrite(&buffer, 1, 4, file);
}

uint32_t
dataPNG::Chunk::calcCRC(){
    uint64_t len = length + 4;
    uint8_t *buf = new uint8_t[len];
    checkpointer(buf);

```

```

uint8_t *ptr = buf;
memcpy(buf, &name, 4);
if(length)
    memcpy(buf + 4, data, length);

uint64_t table[256];
uint64_t crc;
for (int i = 0; i < 256; i++)
{
    crc = i;
    for (int j = 0; j < 8; j++)
        crc = crc & 1 ? (crc >> 1) ^ 0xEDB88320UL : crc >> 1;
    table[i] = crc;
}
crc = 0xFFFFFFFFUL;
while (len--)
    crc = table[(crc ^ *buf++) & 0xFF] ^ (crc >> 8);
delete[] ptr;
return crc ^ 0xFFFFFFFFUL;
}

dataPNG::~dataPNG(){
    for (auto &i : chunks)
        if (i.data){
            delete[] i.data;
            i.data = NULL;
        }
}

int
dataPNG::readPNG(FILE *file){
    Bytef sign[8];
    palette = NULL;
    fread(sign, 1, 8, file);
    if (!memcmp(sign, SIGNATUREPNG, 8)){
        do{
            chunks.push_back(Chunk());
            chunks.back().setFromfile(file);
            if(chunks.back().name == PLTE){
                palette = chunks.back().data;
                palettesize = chunks.back().length;
            }
        }
        while(chunks.back().name != IEND);
        width = reversebyte(*(uint32_t*)&chunks[0].data[0]);
        height = reversebyte(*(uint32_t*)&chunks[0].data[4]);
        depth = *(uint8_t*)&chunks[0].data[8];
        colortype = *(uint8_t*)&chunks[0].data[9];
        for(auto &chunk : chunks){
            if(chunk.crc != chunk.calcCRC()){
                std::cout << "Error: invalid value CRC\n";
                exit(45);
            }
        }

        if(depth != 8){

```

```

        std::cout << "Images with bit depth " << depth << " is
not supported\n";
        exit(46);
    }
    switch(colortype){
    case 6:
        pixelsize = 4;
        break;
    case 3:
        pixelsize = 1;
        break;
    case 2:
        pixelsize = 3;
        break;
    default:
        std::cout << "Color type: " << colortype << " is not
supported\n";
        exit(47);
    }
}
else{
    std::cout << "File is not recognized as a PNG\n";
    exit(44);
}
return 0;
}

void
dataPNG::printdata(){
    printf("Image width: %u\n", width);
    printf("Image height: %u\n", height);
    printf("Image depth: %hhu\n", depth);
    if (colortype == 2){
        printf("Image Colour Type: RGB\n");
    } else if (colortype == 6){
        printf("Image Colour Type: RGBA\n");
    } else {
        printf("Image Colour Type: RGB with palette\n");
    }
}

Image*
dataPNG::decodImage(){
    uint64_t size = sizedata();
    uint8_t *arraybyte;
    uncompressdata(arraybyte, size);
    uint8_t **bitmap = fillimage(arraybyte, size);
    delete[] arraybyte;
    Image* result = new Image;
    checkpointer(result);
    if(colortype == 3){
        fillFromPalette(bitmap);
        pixelsize = 3;
    }
    result->height = height;
    result->width = width;
}

```



```

        result->pixelsize = pixelsize;
        result->bitmap = bitmap;
        return result;
    }

    void
    dataPNG::fillFromPalette(uint8_t **&bitmap){
        uint8_t **newbitmap = new uint8_t*[height];
        checkpointer((void*)newbitmap);
        for(int i = 0; i < height; ++i){
            newbitmap[i] = new uint8_t[width * 3];
            checkpointer(newbitmap[i]);
        }

        for(int i = 0; i < height; ++i)
            for(int j = 0; j < width; ++j)
                if(bitmap[i][j] < palettesize / 3)
                    *(rgb*)&newbitmap[i][j] * 3] =
*(rgb*)&palette[bitmap[i][j] * 3];

        for(int i = 0; i < height; ++i)
            delete[] bitmap[i];
        delete[] bitmap;
        bitmap = newbitmap;
    }

    uint64_t
    dataPNG::sizedata(){
        uint64_t size = 0;
        for(auto &a : chunks)
            if(a.name == IDAT)
                size += a.length;
        return size;
    }

    int
    dataPNG::uncompressdata(uint8_t *&array, uint64_t& size){
        uint8_t *buffer = new uint8_t[size];
        checkpointer(buffer);
        uint8_t *ptr = buffer;
        for(auto &a : chunks)
            if(a.name == IDAT){
                memcpy(ptr, a.data, a.length);
                ptr += a.length;
            }
        uint64_t destlen = width * height * pixelsize + height;
        ptr = new uint8_t[destlen];
        checkpointer(ptr);
        uncompress(ptr, &destlen, buffer, size);
        delete[] buffer;
        array = ptr;
        size = destlen;
        return 0;
    }
}

```

```

uint8_t**
dataPNG::fillimage(uint8_t* array, uint64_t size){
    uint8_t **array2dim = new uint8_t*[height];
    checkpointer(array2dim);
    uint8_t* buffer;
    uint8_t* filters = new uint8_t[height];
    checkpointer(filters);
    for (uint64_t i = 0; i < height; ++i){
        array2dim[i] = array + i * (width * pixelsize + 1);
        filters[i] = array2dim[i][0];
    }

    for(uint64_t i = 0; i < height; ++i){
        buffer = new uint8_t[width * pixelsize];
        checkpointer(buffer);
        memcpy(buffer, array2dim[i] + 1, width * pixelsize);
        array2dim[i] = buffer;
    }

    for(uint64_t i = 0; i < height; ++i)
        for(uint64_t j = 0; j < width * pixelsize; ++j)
            recon(array2dim, filters[i], j, i);

    delete[] filters;
    return array2dim;
}

void
dataPNG::recon(uint8_t **array,      uint8_t  filter,  uint64_t  x,
uint64_t y){
    uint8_t a = 0, b = 0, c = 0;
    if(x >= pixelsize)
        a = array[y][x - pixelsize];
    if(y > 0)
        b = array[y - 1][x];
    if(x >= pixelsize && y > 0)
        c = array[y - 1][x - pixelsize];

    switch(filter){
    case 0:
        break;
    case 1:
        array[y][x] += a;
        break;
    case 2:
        array[y][x] += b;
        break;
    case 3:
        array[y][x] += ((uint16_t)a + (uint16_t)b) / 2;
        break;
    case 4:
        array[y][x] += paethPredictor(a, b, c);
        break;
    }
}

```

```

uint8_t
dataPNG::paethPredictor(int a, int b, int c){
    int p, pa, pb, pc;
    p = a + b - c;
    pa = abs(p - a);
    pb = abs(p - b);
    pc = abs(p - c);
    if (pa <= pb && pa <= pc)
        return a;
    else if (pb <= pc)
        return b;
    else return c;
}

int
dataPNG::writePNG(FILE *file){
    fwrite(SINATUREPNG, 1, 8, file);
    for(auto &chunk : chunks){
        chunk.writeInFile(file);
    }
    return 0;
}

int
dataPNG::updateData(Image &image){
    width = image.width;
    height = image.height;
    pixelsize = image.pixelsize;
    uint64_t size = width * height * pixelsize + height;
    uint8_t *array = new uint8_t[size];
    checkpointer(array);
    for(uint64_t i = 0; i < height; ++i){
        array[i * (width * pixelsize + 1)] = 0;
        memcpy(array + i * (width * pixelsize + 1) + 1,
image.bitmap[i], width * pixelsize);
    }
    compressdata(array, size);
    std::vector<Chunk> uptadechunks;
    uint32_t n = 0;
    while(chunks[n].name != IDAT){
        if(chunks[n].name != PLTE)
            uptadechunks.push_back(chunks[n]);
        ++n;
    }

    for(uint64_t i = 0; i < (size / IDATSIZE); ++i)
        uptadechunks.push_back(Chunk(IDATSIZE, IDAT, array + i *
IDATSIZE));
    if(size % IDATSIZE)
        uptadechunks.push_back(Chunk(size % IDATSIZE, IDAT, array +
(size / IDATSIZE) * IDATSIZE));

    while(n < chunks.size()){
        if (chunks[n].name != IDAT)
            uptadechunks.push_back(chunks[n]);
        ++n;
    }
}

```

```

    }

    chunks = uptadechunks;
    delete[] array;
    *(uint32_t*)&chunks[0].data[0] = reversebyte(width);
    *(uint32_t*)&chunks[0].data[4] = reversebyte(height);
    *(uint8_t*)&chunks[0].data[8] = 8;
    *(uint8_t*)&chunks[0].data[9] = (pixelsize == 4) ? 6 : 2;
    chunks[0].crc = chunks[0].calcCRC();
    return 0;
}

int
dataPNG::compressdata(uint8_t *&array, uint64_t &size){
    uint64_t destlen = compressBound(size);
    uint8_t *dest = new uint8_t[destlen];
    checkpointer(dest);
    compress(dest, &destlen, array, size);
    delete[] array;
    size = destlen;
    array = dest;
    return 0;
}

```

Файл image.hpp

```

#pragma once
#include<iostream>

#pragma pack(push, 1)
struct rgb{
    uint8_t r, g, b;
    rgb();
    rgb(int, int, int);
    int toint();
};
#pragma pack(pop)

class Image{
public:
    uint32_t height, width;
    uint8_t pixelsize;
    uint8_t **bitmap;
    void setpixel(int, int, rgb &);
    rgb *getpixel(int, int);
    bool checkCoordinats(int, int);
    ~Image();
};

```

Файл image.cpp

```

#include "image.hpp"

Image::~Image(){

```

```

        if(bitmap){
            for(int i = 0; i < height; ++i)
                delete[] bitmap[i];
            delete[] bitmap;
        }
    }

    void
    Image::setpixel(int x, int y, rgb &color){
        if(checkCoordinats(x, y)){
            *getpixel(x, y) = color;
        }
    }

    rgb*
    Image::getpixel(int x, int y){
        if(checkCoordinats(x, y))
            return (rgb*)&bitmap[y][x * pixelsize];
        return NULL;
    }

    bool
    Image::checkCoordinats(int x, int y){
        return (x >= 0 && x < width && y >= 0 && y < height);
    }

    int
    rgb::toint(){
        return ((int(r) << 16) + (int(g) << 8) + int(b));
    }

    rgb::rgb(int x, int y, int z){
        r = x;
        g = y;
        b = z;
    }

    rgb::rgb(){
        r = 0;
        g = 0;
        b = 0;
    }
}

```

Файл input\_arguments.hpp

```

#pragma once
#include <iostream>
#include <cstring>
#include <getopt.h>

#include "image.hpp"

class arguments{

```

```

public:
    bool ishelp;
    bool isinfo;
    bool isfill;
    int number;
    int x1;
    int x2;
    int y1;
    int y2;
    int length;
    int thickness;
    int type_exchange;
    int color[3];
    int fill_color[3];
    char* input_path;
    char* output_path;

    arguments();
    ~arguments();
};

```

```
arguments *getarguments(int argc, char** argv);
```

Файл input\_arguments.cpp

```
#include "input_arguments.hpp"
```

```

arguments::arguments(){
    ishelp = false;
    isinfo = false;
    isfill = false;
    number = 0;
    x1 = -1;
    x2 = -1;
    y1 = -1;
    y2 = -1;
    length = 0;
    thickness = 0;
    type_exchange = 0;
    color[0] = -1;
    fill_color[0] = -1;
    input_path = NULL;
    output_path = strdup("out.png");
    if (output_path == NULL){
        std::cout << "Error: not enough memory\n";
        exit(48);
    }
}

arguments::~~arguments(){
    if (input_path)
        free(input_path);
    if (output_path)
        free(output_path);
}

```

```

}

arguments *getarguments(int argc, char** argv){
    arguments *args = new arguments;
    if (args == NULL){
        std::cout << "Error: not enough memory\n";
        exit(48);
    }
    args->input_path = strdup(argv[argc - 1]);
    if (args->input_path == NULL){
        std::cout << "Error: not enough memory\n";
        exit(48);
    }
    const char* short_options = "hpSEFI:l:r:s:t:c:C:fe:o:i:R";
    opterr = 0;
    const struct option long_options[] ={
        {"help", no_argument, NULL, 'h'},
        {"info", no_argument, NULL, 'p'},
        {"square", no_argument, NULL, 'S'},
        {"exchange", no_argument, NULL, 'E'},
        {"freq_color", no_argument, NULL, 'F'},
        {"inverse", no_argument, NULL, 'I'},
        {"left_up", required_argument, NULL, 'l'},
        {"right_down", required_argument, NULL, 'r'},
        {"side_size", required_argument, NULL, 's'},
        {"thickness", required_argument, NULL, 't'},
        {"color", required_argument, NULL, 'c'},
        {"fill_color", required_argument, NULL, 'C'},
        {"fill", no_argument, NULL, 'f'},
        {"exchange_type", required_argument, NULL, 'e' },
        {"output", required_argument, NULL, 'o' },
        {"input", required_argument, NULL, 'i' },
        {"rhombus", no_argument, NULL, 'R' },
        {NULL, 0, NULL, 0 }
    };
    int n, opt;
    while ((opt = getopt_long(argc, argv, short_options,
long_options, NULL)) != -1){
        switch (opt){
            case 'h': {
                args->ishelp = true;
                break;
            }
            case 'p': {
                args->isinfo = true;
                break;
            }
            case 'S': {
                if(args->number){
                    std::cout << "More then one command entered\n";
                    exit(41);
                }
                args->number = 1;
                break;
            }
            case 'E': {

```

```

        if(args->number){
            std::cout << "More then one command entered\n";
            exit(41);
        }
        args->number = 2;
        break;
    }
    case 'F': {
        if(args->number){
            std::cout << "More then one command entered\n";
            exit(41);
        }
        args->number = 3;
        break;
    }
    case 'I': {
        if(args->number){
            std::cout << "More then one command entered\n";
            exit(41);
        }
        args->number = 4;
        break;
    }
    case 'R': {
        if(args->number){
            std::cout << "More then one command entered\n";
            exit(41);
        }
        args->number = 5;
        break;
    }
    case 'l': {
        n = sscanf(optarg, "%d.%d", &args->x1, &args->y1);
        if (n != 2){
            std::cout << "Invalid point format\n";
            exit(41);
        }
        break;
    }
    case 'r': {
        n = sscanf(optarg, "%d.%d", &args->x2, &args->y2);
        if (n != 2){
            std::cout << "Invalid point format\n";
            exit(41);
        }
        break;
    }
    case 's': {
        n = sscanf(optarg, "%d", &args->length);
        if (n != 1 || args->length <= 0){
            std::cout << "Invalid side size format\n";
            exit(41);
        }
        if (args->length <= 0){
            std::cout << "Invalid side size value\n";
            exit(42);
        }
    }

```



```

        }
        break;
    }
    case 't': {
        n = sscanf(optarg, "%d", &args->thickness);
        if (n != 1){
            std::cout << "Invalid thickness format\n";
            exit(41);
        }
        if (args->thickness <= 0){
            std::cout << "Invalid thickness value\n";
            exit(42);
        }
    }
    break;
    case 'c': {
        n = sscanf(optarg, "%d.%d.%d", &args->color[0], &args->
color[1], &args->color[2]);
        if (n != 3){
            std::cout << "Invalid color format\n";
            exit(41);
        }
        if(args->color[0] < 0 || args->color[0] > 255 || args->
color[1] < 0 || args->color[1] > 255 || args->color[2] < 0 || args->
color[2] > 255){
            std::cout << "Invalid color value\n";
            exit(42);
        }
    }
    break;
    case 'C': {
        n = sscanf(optarg, "%d.%d.%d", &args->fill_color[0],
&args->fill_color[1], &args->fill_color[2]);
        if (n != 3){
            std::cout << "Invalid fill color format\n";
            exit(41);
        }
        if(args->fill_color[0] < 0 || args->fill_color[0] > 255
|| args->fill_color[1] < 0 || args->fill_color[1] > 255 || args->
fill_color[2] < 0 || args->fill_color[2] > 255){
            std::cout << "Invalid fill color value\n";
            exit(42);
        }
    }
    break;
    case 'f': {
        args->isfill = true;
        break;
    }
    case 'e': {
        if(strcmp(optarg, "clockwise") == 0)
            args->type_exchange = 1;
        else if(strcmp(optarg, "counterclockwise") == 0)
            args->type_exchange = 2;
        else if(strcmp(optarg, "diagonals") == 0)
            args->type_exchange = 3;
    }
}

```

```

        else{
            std::cout << "Invalid exchange type\n";
            exit(42);
        }
        break;
    }
    case 'o': {
        free(args->output_path);
        args->output_path = strdup(optarg);
        if (args->output_path == NULL){
            std::cout << "Error: not enough memory\n";
            exit(48);
        }
        break;
    }
    case 'i': {
        free(args->input_path);
        args->input_path = strdup(optarg);
        if (args->input_path == NULL){
            std::cout << "Error: not enough memory\n";
            exit(48);
        }
        break;
    }
    default: {
        std::cout << "Unknown flag\n";
        exit(43);
    }
}
}

if(!strcmp(args->input_path, args->output_path)){
    std::cout << "Input and output cannot have the same path\n";
    exit(49);
}
return args;
}

```

Файл square.hpp

```

#pragma once
#include "image.hpp"

void drawSquare(Image *, int, int, int, int, rgb);
void drawSquare(Image *, int, int, int, int, rgb, rgb);

```

Файл square.cpp

```

#include "square.hpp"

void drawSquare(Image *image, int x, int y, int l, int thickness,
rgb color){
    for(int k = 0; k < thickness; ++k){
        for(int i = x + k; i < x + l - k; ++i){
            image->setpixel(i, y + k, color);
        }
    }
}

```

```

        image->setpixel(i, y + l - k - 1, color);
    }
    for(int j = y + k; j < y + l - k; ++j){
        image->setpixel(x + k, j, color);
        image->setpixel(x + l - k - 1, j, color);
    }
}

void drawSquare(Image *image, int x, int y, int l, int thickness,
rgb color, rgb fill){
    x -= thickness / 2;
    y -= thickness / 2;
    drawSquare(image, x, y, l, thickness, color);
    x += thickness;
    y += thickness;
    l = (l >= thickness)? l - 2 * thickness: 0;
    for(int i = x; i < x + l; ++i)
        for(int j = y; j < y + l; ++j)
            image->setpixel(i, j, fill);
}

```

Файл exchange.hpp

```

#pragma once
#include <cstring>
#include "image.hpp"

void exchange(Image *, int, int, int, int, int);

void copyrect(uint8_t **, uint8_t **, int, int, int, int, int,
int);

uint8_t **newtable(int, int);

```

Файл exchange.cpp

```

#include "exchange.hpp"

void exchange(Image *image, int x1 , int y1, int x2 , int y2, int
type){
    int size = image->pixelsize;
    x1 = (x1 < 0)? 0 : x1;
    y1 = (y1 < 0)? 0 : y1;
    x2 = (x2 > image->width)? image->width : x2;
    y2 = (y2 > image->height)? image->height : y2;
    x2 -= (x2 - x1) % 2;
    y2 -= (y2 - y1) % 2;

    int x = x2 - x1;
    int y = y2 - y1;
    if (x <= 1 || y <= 1)
        return;
}

```

```

uint8_t **buffer = newtable(y, x * size);
copyrect(buffer, image->bitmap, 0, 0, x1 * size, y1, x * size,
y);

x /= 2;
y /= 2;

switch(type){
    case 1:
        copyrect(image->bitmap, buffer, x1 * size, y1, 0, y, x *
size, y);
        copyrect(image->bitmap, buffer, (x1 + x) * size, y1, 0,
0, x * size, y);
        copyrect(image->bitmap, buffer, (x1 + x) * size, y1 + y,
x * size, 0, x * size, y);
        copyrect(image->bitmap, buffer, x1 * size, y1 + y, x *
size, y, x * size, y);
        break;
    case 2:
        copyrect(image->bitmap, buffer, x1 * size, y1, x * size,
0, x * size, y);
        copyrect(image->bitmap, buffer, (x1 + x) * size, y1, x *
size, y, x * size, y);
        copyrect(image->bitmap, buffer, (x1 + x) * size, y1 + y,
0, y, x * size, y);
        copyrect(image->bitmap, buffer, x1 * size, y1 + y, 0, 0,
x * size, y);
        break;
    case 3:
        copyrect(image->bitmap, buffer, x1 * size, y1, x * size,
y, x * size, y);
        copyrect(image->bitmap, buffer, (x1 + x) * size, y1 + y,
0, 0, x * size, y);
        copyrect(image->bitmap, buffer, (x1 + x) * size, y1, 0,
y, x * size, y);
        copyrect(image->bitmap, buffer, x1 * size, y1 + y, x *
size, 0, x * size, y);
        break;
}

for(int i = 0; i < y * 2; ++i)
    delete[] buffer[i];
delete[] buffer;
}

void copyrect(uint8_t **dest, uint8_t **src, int x1 , int y1, int
x2 , int y2, int x, int y){
    for(int i = 0; i < y; ++i)
        memcpy(dest[y1 + i] + x1, src[y2 + i] + x2, x);
}

uint8_t **newtable(int height, int width){

```

```

uint8_t **res = new uint8_t*[height];
if (res == NULL){
    std::cout << "Error: not enough memory\n";
    exit(48);
}
for(int i = 0; i < height; ++i){
    res[i] = new uint8_t[width];
    if (res[i] == NULL){
        std::cout << "Error: not enough memory\n";
        exit(48);
    }
}
return res;
}

```

Файл freqcolor.hpp

```

#pragma once
#include<map>
#include "image.hpp"

```

```

void freqcolor(Image *, rgb);

```

Файл freqcolor.cpp

```

#include "freqcolor.hpp"

```

```

void freqcolor(Image *image, rgb color){
    std::map<int, int> colorstree;
    for(int i = 0; i < image->height; ++i)
        for(int j = 0; j < image->width; ++j){
            int key = ((rgb*)&image->bitmap[i][j] * image->pixelsize)]->toint();
            colorstree[key] += 1;
        }
    int maxcount = 0, col = 0;
    for(auto &val : colorstree){
        if(val.second > maxcount){
            col = val.first;
            maxcount = val.second;
        }
    }
    for(int i = 0; i < image->height; ++i)
        for(int j = 0; j < image->width; ++j)
            if(((rgb*)&image->bitmap[i][j] * image->pixelsize)]->toint() == col)
                *(rgb*)&image->bitmap[i][j] * image->pixelsize] = color;
}

```

Файл inverse.hpp

```
#pragma once
#include "image.hpp"

void inverserect(Image *, int, int, int, int);
void inversepixel(rgb *);
```

Файл inverse.cpp

```
#include "inverse.hpp"

void
inverserect(Image *image, int x1, int y1, int x2, int y2){
    for(int i = x1; i < x2; ++i)
        for(int j = y1; j < y2; ++j)
            inversepixel(image->getpixel(i, j));
}

void
inversepixel(rgb *pixel){
    if(pixel){
        pixel->r = ~pixel->r;
        pixel->g = ~pixel->g;
        pixel->b = ~pixel->b;
    }
}
```