

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

КУРСОВАЯ РАБОТА
по дисциплине «Программирование»
Тема: Обработка изображений на языке C++

Студент гр. 0382

Диденко Д.В.

Преподаватель

Берленко Т.А.

Санкт-Петербург

2021

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Диденко Д.В.

Группа 0382

Тема работы: Обработка изображений на языке C++

Исходные данные:

Программе на вход подается изображение в PNG формате. Необходимо его считать, преобразовать и сохранить измененную копию. Управление операциями осуществляется через интерфейс командной строки с использованием библиотеки getopt.h.

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания: 16.05.2021

Дата сдачи реферата: 17.05.2021

Дата защиты реферата: 18.05.2021

Студент	_____	Диденко Д.В.
---------	-------	--------------

Преподаватель	_____	Берленко Т.А.
---------------	-------	---------------

АННОТАЦИЯ

В курсовой работе была реализована обработка PNG изображения с типом цвета RGBA, для этого был создан класс, в котором определены необходимые члены-функции и члены-переменные. Для чтения и записи изображения использовалась библиотека `libpng`. В программе реализован интерфейс командной строки с помощью библиотеки `getopt.h`. При вводе некорректных команд или неправильных начальных данных изменения изображения пользователю выводятся соответствующие ошибки. Для сборки программы используется `Makefile`.

СОДЕРЖАНИЕ

Введение.	5
1. Цель и задание работы.	6
1.1. Цель работы.	6
1.2. Задание работы.	6
2. Ход реализации работы.	8
2.1. Создание класса.	8
2.2. Считывание и хранение изображения в бинарном виде.	9
2.3. Решение задачи 1.	9
2.4. Решение задачи 2.	10
2.5. Решение задачи 3.	10
2.6. Решение задачи 4.	10
2.7. Сохранение нового изображения.	11
2.8. Реализация CLI.	11
3. Тестирование.	12
Заключение.	16
Список использованных источников.	17
Приложение А. Исходный код программы.	18

ВВЕДЕНИЕ

Требуется создать программу, производящую выбранную пользователем обработку изображения.

Программа реализована для операционных систем на базе Linux. Разработка велась на базе операционной системы Ubuntu Linux в IDE Clion. Компиляция и линковка производилась с помощью Makefile.

В результате была создана программа, считывающая исходное PNG изображение в бинарном виде, сохраняющая всю информацию в структуру, обрабатывающая бинарные данные функцией, выбранной пользователем и сохраняющая измененное изображение в PNG формате. Также производятся действия по очистке динамически выделенной памяти и корректному завершению работы.

1. ЦЕЛЬ И ЗАДАНИЕ РАБОТЫ

1.1. Цель работы.

Цель работы: создать программу, производящую выбранную пользователем обработку изображения.

Для достижения поставленной цели требуется решить следующие задачи:

- Реализовать бинарное считывание изображения.
- Реализовать функции, решающие обозначенные задачи.
- Создать копию изображения с изменениями.
- Реализовать CLI.

1.2. Задание курсовой работы.

Вариант 15

- Программа должна иметь CLI или GUI.
- Общие сведения
- Формат картинки PNG.
- Файл всегда соответствует формату PNG.
- 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 реализовывать следующий функционал по обработке PNG-файла:

1. Отражение заданной области. Этот функционал определяется:

- выбором оси относительно которой отражать (горизонтальная или вертикальная)
- Координатами левого верхнего угла области
- Координатами правого нижнего угла области

2. Рисование пентаграммы в круге. Пентаграмма определяется:

- либо координатами левого верхнего и правого нижнего угла квадрата, в который вписана окружность пентаграммы, либо координатами ее центра и радиусом окружности
- толщиной линий и окружности
- цветом линий и окружности

3. 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.

4. Рисование правильного шестиугольника. Шестиугольник определяется:

- Либо координатами левого верхнего и правого нижнего угла квадрата, в который он вписан, либо координатами его центра и радиусом в который он вписан.
- Толщиной линий.
- Цветом линий.
- Шестиугольник может быть залит или нет.
- Цветом которым залит шестиугольник, если пользователем выбран залитый.

2. РЕАЛИЗАЦИЯ РАБОТЫ.

2.1. Создание класса.

Для работы с изображением создается класс `images`, в котором определены все необходимые переменные- и функции-члены.

Приватные члены-переменные:

- `char header[8]`
- `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 point[6][2] = {{0,0},{0,0},{-1,-1},{0,0},{0,0},{0,0}};`
- `struct Png* image = new struct Png;`
- `int RGBA[4] = {-1,-1,-1,-1};`
- `int RGBA_Fill[4] = {-1,-1,-1,-1};`
- `int center[2] = {-1,-1};`
- `int bold = 0;`
- `int fill = 0;`
- `int R = 0;`
- `int Ax = -1;`

Публичные члены-функции:

- `~images()` - деструктор
- `void read_png_file(char *file_name);`
- `void write_png_file(char *file_name);`
- `void drawpixel(int x,int y);`

- void drawpixel_Fill(int x,int y);
- void drawLine(int x1, int y1, int x2, int y2);
- void drawLine_Fill(int x1, int y1, int x2, int y2);
- void reflection();
- void paint_pentogram();
- void paint_rectangle();
- void paint_hexagon();
- int is_error();
- int is_error_R();
- int is_error_Color();
- int is_error_Color_Fill();
- struct Png* get_info();
- void set_center(int x1, int y1);
- void set_point(int N, int x1, int y1);
- void set_axis(int a);
- void set_radius(int a);
- void set_color(int i, int c);
- void set_color_fill(int i, int c);
- void set_bold(int b);
- void set_fill(int f);

2.2. Считывание и хранение изображения в бинарном виде.

Считывание изображения производится в методе read_png_file(char *file_name) класса images с помощью функций библиотеки libpng из файла с названием file_name, данные хранятся в структуре Png класса и массиве header. Если изображение не удастся считать, об этом выводится соответствующая ошибка.

2.3. Решение задачи 1.

Для решения этой задачи вызывается метод reflection(), использующий точки point[0][0] и point[0][1], point[2][0] и point[2][1] (левый верхний и правый

нижний углы) и ось отражения Ax. Возможные ошибки обрабатываются и выводятся в консоль.

2.4. Решение задачи 2.

Для решения этой задачи вызывается метод `paint_pentogram()`, использующий либо точки `point[0][0]` и `point[0][1]`, `point[2][0]` и `point[2][1]` (левый верхний и правый нижний углы квадрата, в который вписана окружность с пентограммой), либо `center[2]` и `R` (координаты центра и радиус окружности). Для рисования круга и линий используется алгоритм Брезенхема. Возможные ошибки обрабатываются и выводятся в консоль.

2.5. Решение задачи 3.

Для решения этой задачи вызывается метод `paint_rectangle()`, использующий точки `point[0][0]` и `point[0][1]`, `point[2][0]` и `point[2][1]` (левый верхний и правый нижний углы прямоугольника), `bold` — жирность границы, `RGBA[4]` — цвет границы, `fill` — залит ли прямоугольник, `RGBA_Fill[4]` — цвет заливки, цвета устанавливаются в диапазоне 0 - 255. Возможные ошибки обрабатываются и выводятся в консоль.

2.5. Решение задачи 4.

Для решения этой задачи вызывается метод `paint_hexagon()`, использующий либо точки `point[0][0]` и `point[0][1]`, `point[2][0]` и `point[2][1]` (левый верхний и правый нижний углы квадрата, в который вписана окружность с шестиугольником), либо `center[2]` и `R` (координаты центра и радиус окружности) `bold` — жирность границы, `RGBA[4]` — цвет границы, `fill` — залит ли прямоугольник, `RGBA_Fill[4]` — цвет заливки, цвета устанавливаются в диапазоне 0 - 255. Для рисования пентограммы используется алгоритм Евклида для нахождения остальных точек шестиугольника и алгоритм Брезенхема для их соединения. Возможные ошибки обрабатываются и выводятся в консоль.

2.7. Сохранение нового изображения.

Считывание изображения производится в методе `write_png_file(char *file_name)` класса `images` с помощью функций библиотеки `libpng`, данные записываются в файл с названием `file_name`. Если изображение не удастся записать, об этом выводится соответствующая ошибка.

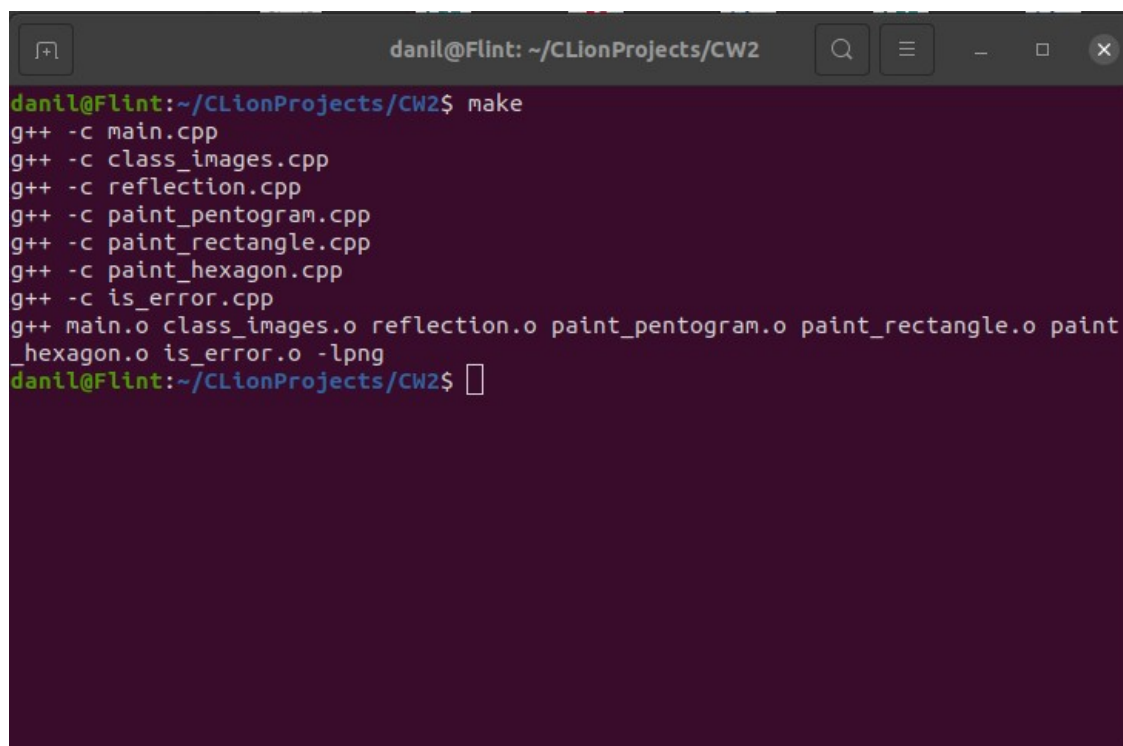
2.8. Реализация CLI.

Управление программой осуществляется посредством аргументов командной строки. Реализовано с использованием библиотеки `getopt.h`.

Список команд показывается с использованием ключа `—help` или `-H`. Каждая команда вызывает соответствующий метод класса и через него устанавливает введенные как аргументы установки. При запуске программы после имени запускаемого файла обязательно должно стоять имя обрабатываемого изображения.

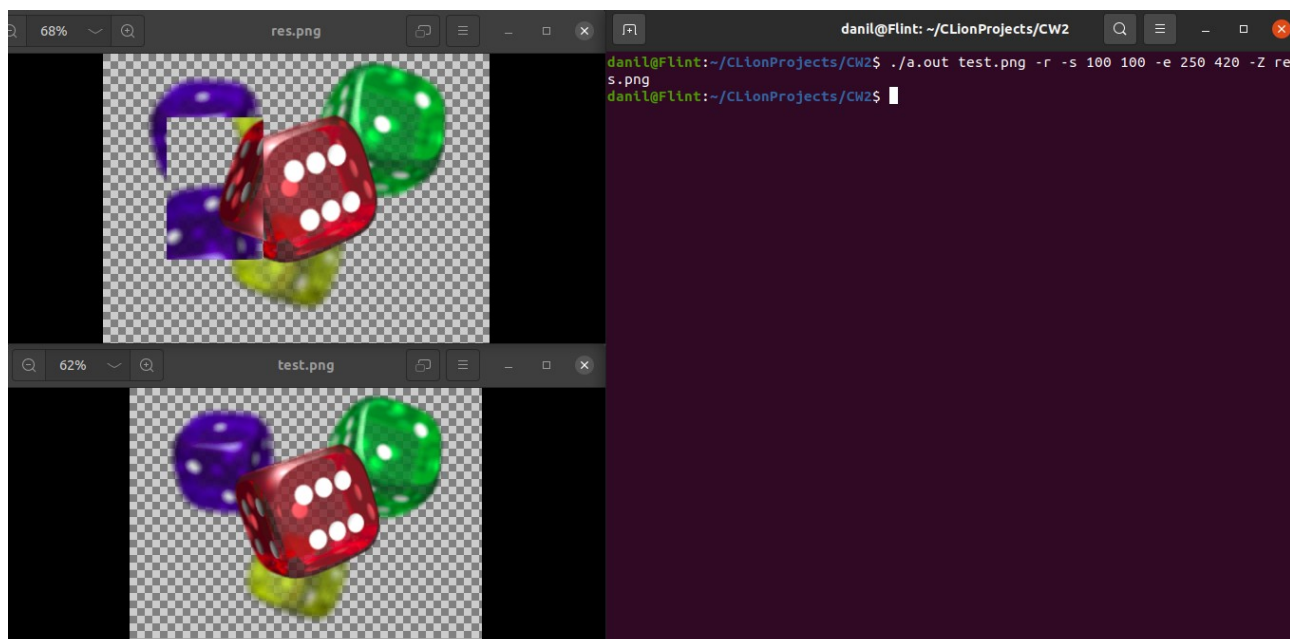
3. ТЕСТИРОВАНИЕ

1. Работа Makefile.

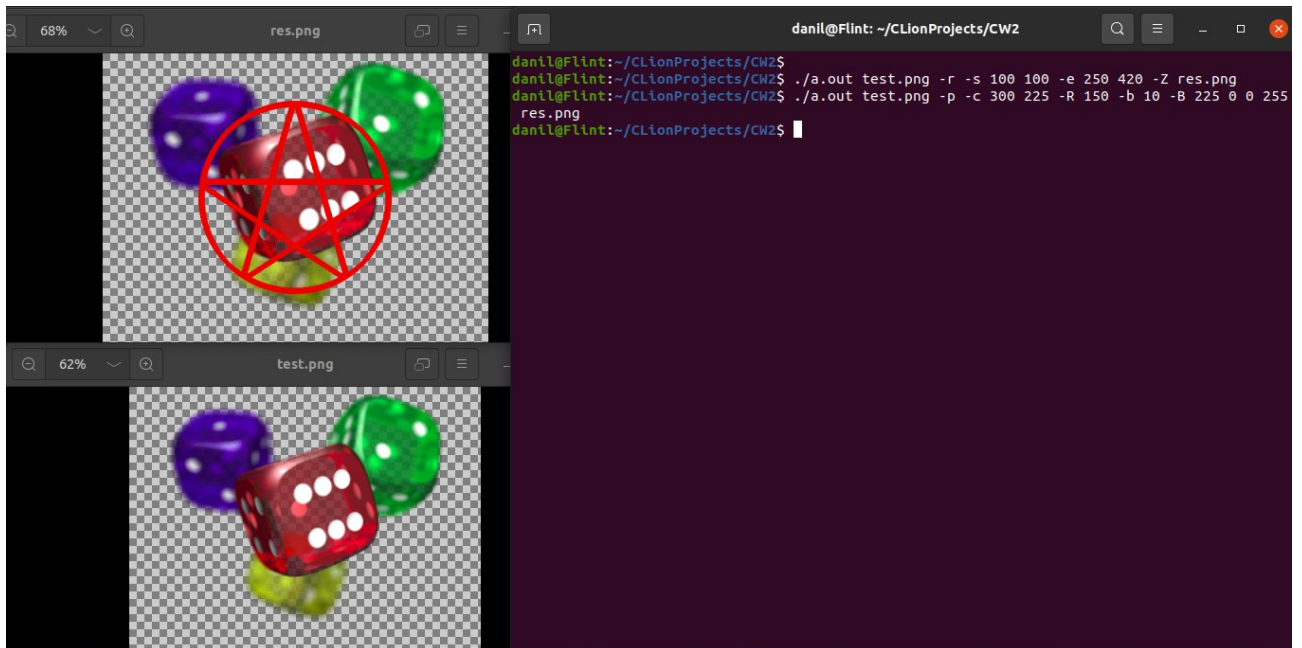


```
danil@Flint:~/CLionProjects/CW2$ make
g++ -c main.cpp
g++ -c class_images.cpp
g++ -c reflection.cpp
g++ -c paint_pentogram.cpp
g++ -c paint_rectangle.cpp
g++ -c paint_hexagon.cpp
g++ -c is_error.cpp
g++ main.o class_images.o reflection.o paint_pentogram.o paint_rectangle.o paint_hexagon.o is_error.o -lpng
danil@Flint:~/CLionProjects/CW2$
```

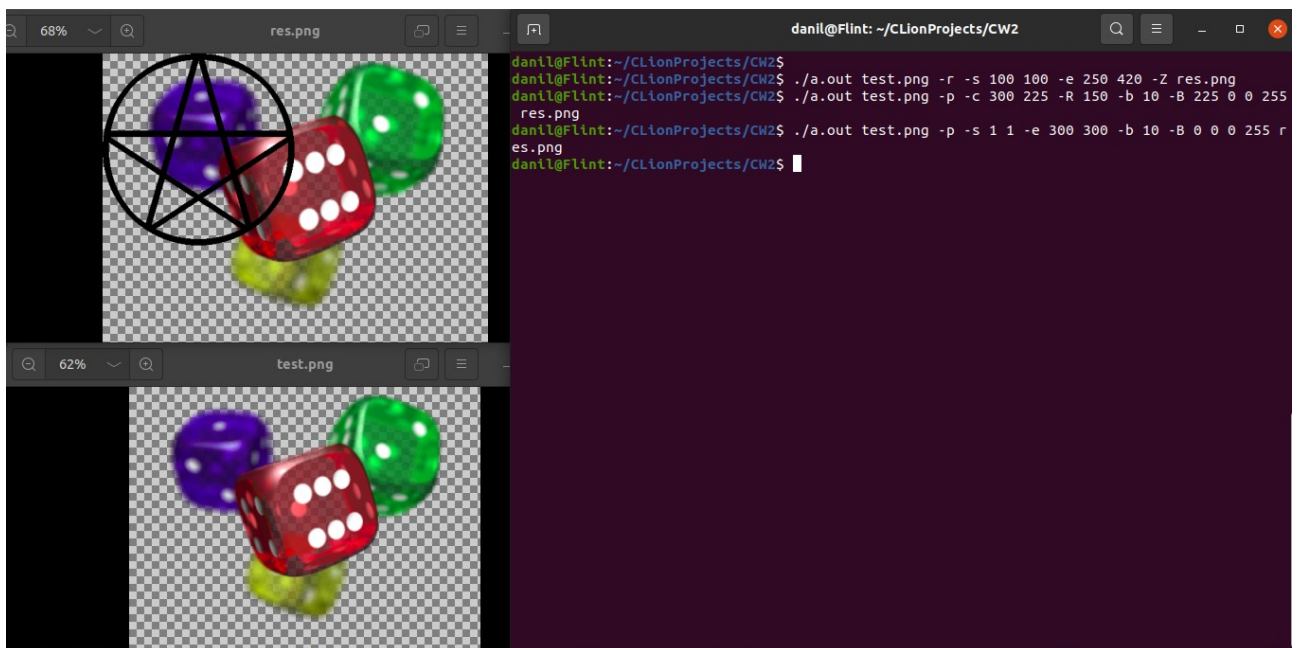
2. Пример отражения заданной области.



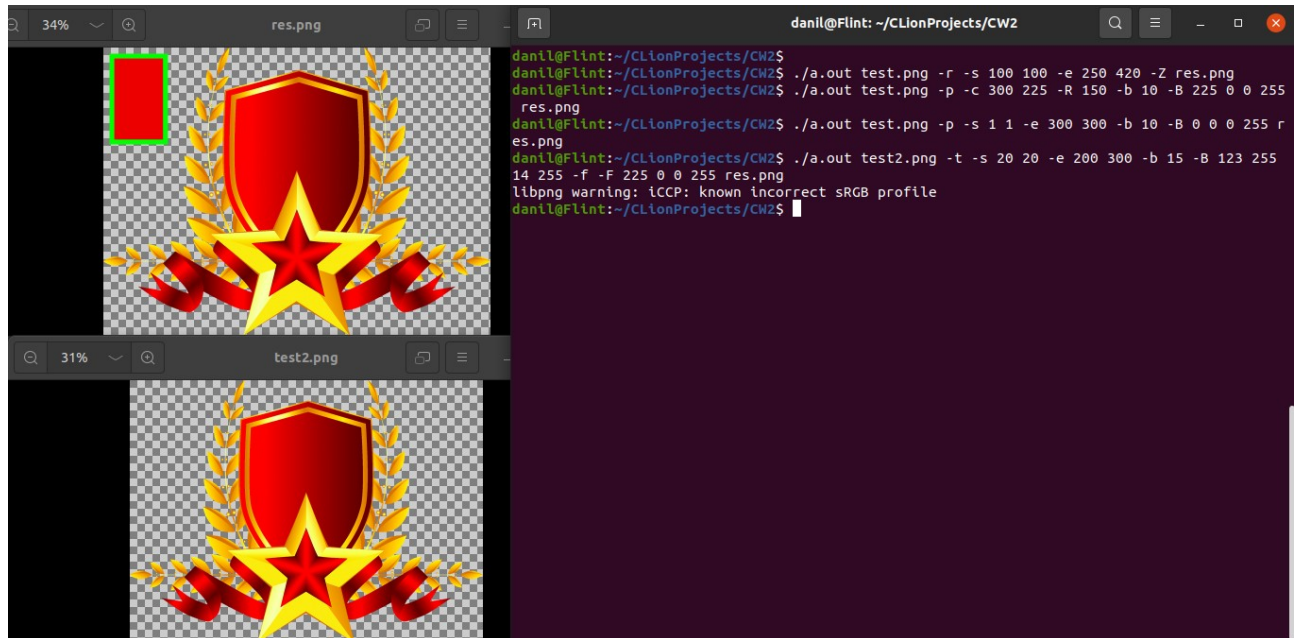
3.1. Пример рисования пентограммы с использованием центра и радиуса.



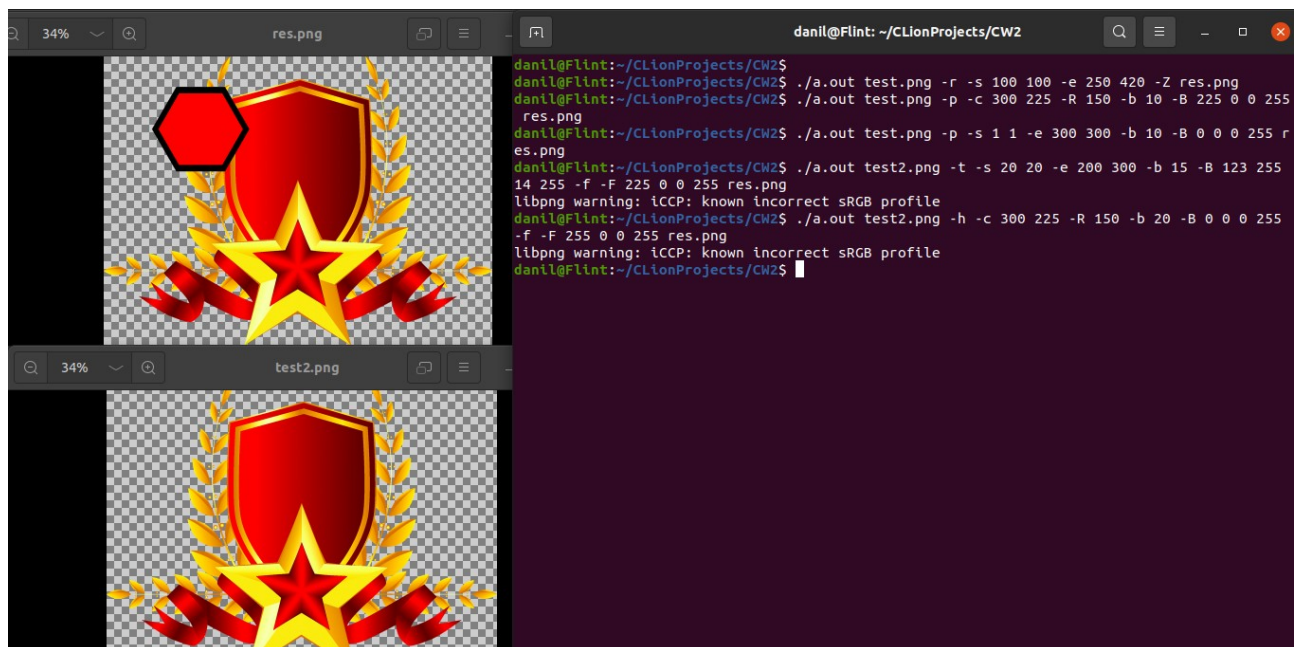
3.2. Пример рисования пентограммы с использованием левого верхнего и правого нижнего углов.



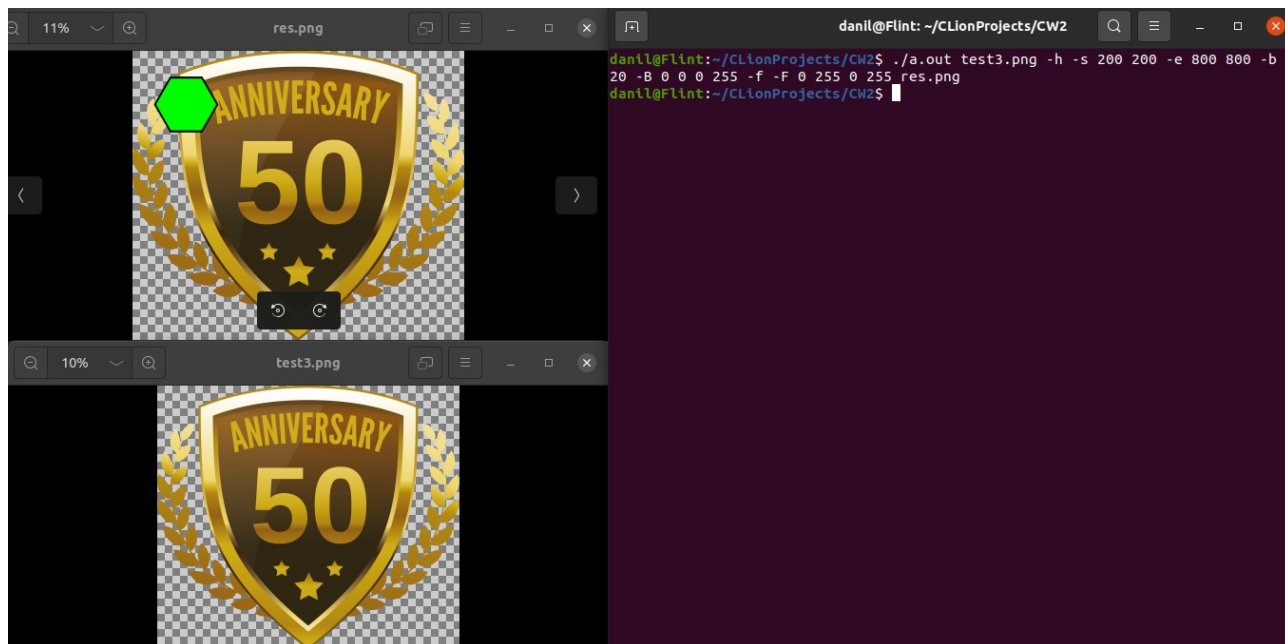
4. Пример рисования прямоугольника.



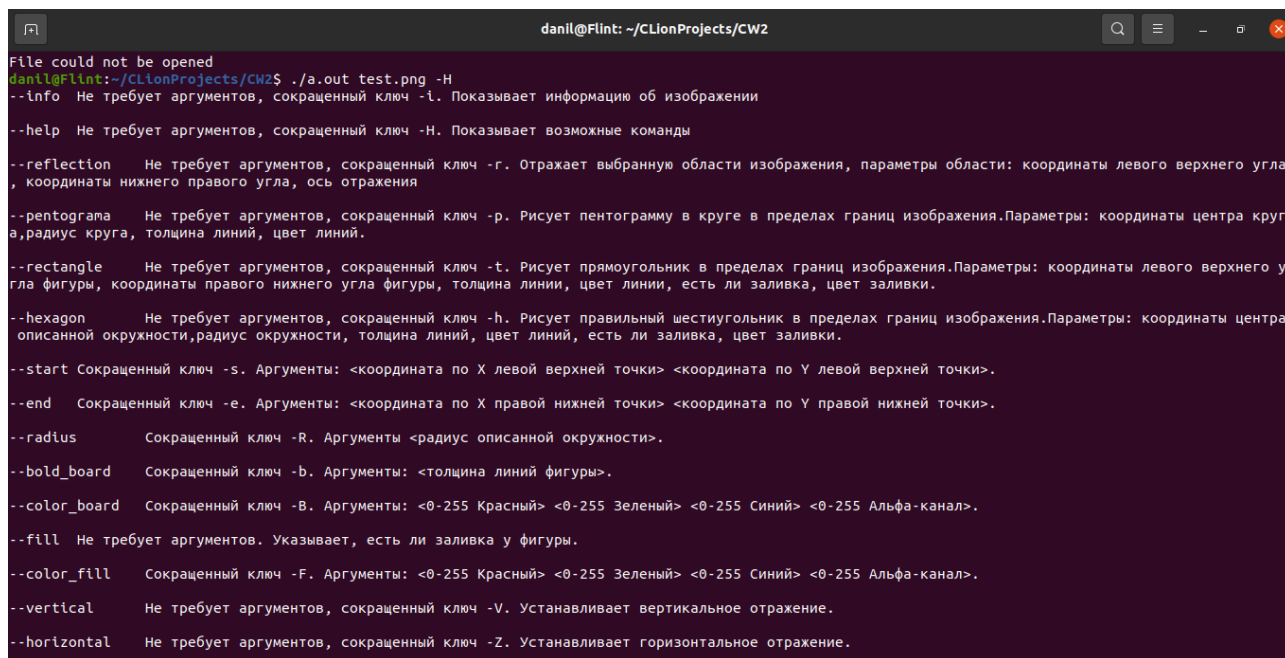
5.1. Пример рисования шестиугольника с центром и радиусом.



5.2. Пример рисования шестиугольника с левым верхним и правым нижним углами описанного квадрата.



6. Пример вывода справки через CLI.



ЗАКЛЮЧЕНИЕ

Разработана стабильная программа, считывающая исходное PNG изображение с цветовым типом RGBA в бинарном виде, сохраняющая всю информацию в структуру, обрабатывающая бинарные данные функцией, выбранной пользователем и сохраняющая измененное изображение в PNG формате. Взаимодействие между пользователем и программой происходит через CLI.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. <https://www.cplusplus.com/>
2. <https://habr.com/ru/post/130472/#:~:text=%D0%9C%D0%B8%D0%BD%D0%B8%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9%20PNG&text=%D0%A8%D0%B8%D1%80%D0%B8%D0%BD%D0%B0%2C%204%20%D0%B1%D0%B0%D0%B9%D1%82%D0%B0,%D0%B0%D0%BB%D1%8C%D1%84%D0%B0%2D%D0%BA%D0%B0%D0%BD%D0%B0%D0%BB>
3. <http://www.libpng.org/pub/png/libpng-1.2.5-manual.html>
4. https://www.gnu.org/software/libc/manual/html_node/Getopt.html

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ.

1. Файл main.cpp

```
#include <cstdlib>
#include <iostream>
#include "class_images.h"
#include <getopt.h>
#include <unistd.h>
#include <cstring>

int main(int argc, char **argv) {
    static int function_flag = 0;
    char* name_out_file = argv[argc-1];
    images image;
    image.read_png_file(argv[1]);
    int c;
    while (1)
    {
        static struct option long_options[] =
            {
                {"help",no_argument,0, 'H'},
                {"info",no_argument,0, 'i'},
                {"reflection",no_argument,0, 'r'},
                {"pentograma",no_argument,0, 'p'},
                {"rectangle",no_argument,0, 't'},
                {"hexagon",no_argument,0, 'h'},
                {"center",required_argument,0, 'c'},
                {"start",required_argument,0, 's'},
                {"end", required_argument,0, 'e'},
                {"radius",required_argument,0, 'R'},
                {"bold_board",required_argument,0, 'b'},
                {"color_board",required_argument,0, 'B'},
                {"fill",no_argument,0, 'f'},
                {"color_fill",required_argument,0, 'F'},
                {"vertical", no_argument,0, 'V'},
                {"horizontal", no_argument,0, 'Z'},
                {"save", required_argument,0, 'S'},
                {0, 0, 0, 0}
            };
        static int option_index = 0;
        c = getopt_long (argc, argv, "iHrpthc:s:e:R:b:B:fF:ZVS:",
long_options, &option_index);
        if (c == -1)
            break;
        switch (c)
        {
            case 'H':
                function_flag = 0;
                std::cout << "--info\the требует аргументов, сокращенный
ключ -i. Показывает информацию об изображении\n\n";
                std::cout << "--help\the требует аргументов, сокращенный
ключ -H. Показывает возможные команды\n\n";
                std::cout << "--reflection\the требует аргументов,
сокращенный ключ -r. Отражает выбранную области изображения,"
```

```

        " параметры области: координаты левого
верхнего угла, координаты нижнего правого угла, ось отражения\n\n";
        std::cout << "--pentograma\tНе требует аргументов,
сокращенный ключ -p. Рисует пентограмму в круге в пределах границ
изображения."

        "Параметры: координаты центра круга, радиус
круга, толщина линий, цвет линий.\n\n";
        std::cout << "--rectangle\tНе требует аргументов,
сокращенный ключ -t. Рисует прямоугольник в пределах границ изображения."
        "Параметры: координаты левого верхнего угла
фигуры, координаты правого нижнего угла фигуры, толщина линии, цвет
линии, есть ли заливка, цвет заливки.\n\n";
        std::cout << "--hexagon\tНе требует аргументов,
сокращенный ключ -h. Рисует правильный шестиугольник в пределах границ
изображения."

        "Параметры: координаты центра описанной
окружности, радиус окружности, толщина линий, цвет линий, есть ли заливка,
цвет заливки.\n\n";
        std::cout << "--start\tСокращенный ключ -s. Аргументы:
<координата по X левой верхней точки> <координата по Y левой верхней
точки>.\n\n";
        std::cout << "--end\tСокращенный ключ -e. Аргументы:
<координата по X правой нижней точки> <координата по Y правой нижней
точки>.\n\n";
        std::cout << "--radius\tСокращенный ключ -R. Аргументы
<радиус описанной окружности>.\n\n";
        std::cout << "--bold_board\tСокращенный ключ -b.
Аргументы: <толщина линий фигуры>.\n\n";
        std::cout << "--color_board\tСокращенный ключ -B.
Аргументы: <0-255 Красный> <0-255 Зеленый> <0-255 Синий> <0-255 Альфа-
канал>.\n\n";
        std::cout << "--fill\tНе требует аргументов. Указывает,
есть ли заливка у фигуры.\n\n";
        std::cout << "--color_fill\tСокращенный ключ -F.
Аргументы: <0-255 Красный> <0-255 Зеленый> <0-255 Синий> <0-255 Альфа-
канал>.\n\n";
        std::cout << "--vertical\tНе требует аргументов,
сокращенный ключ -V. Устанавливает вертикальное отражение.\n\n";
        std::cout << "--horizontal\tНе требует аргументов,
сокращенный ключ -Z. Устанавливает горизонтальное отражение.\n\n";
        std::cout << "--save\tСокращенный ключ -S. Аргументы: имя
сохраняемого файла.\n\n";
        break;
    case 'i':
        std::cout << "Ширина в пикселях: " << image.get_info()-
>width << "\n";
        std::cout << "Высота в пикселях: " << image.get_info()-
>height << "\n";
        printf("Тип цвета: %u\n", image.get_info()->color_type);
        printf("Глубина цвета: %u бит\n", image.get_info()-
>bit_depth);
        break;
    case 'r':
        function_flag = 1;
        break;
    case 'p':
        function_flag = 2;

```

```

        break;
    case 't':
        function_flag = 3;
        break;
    case 'h':
        function_flag = 4;
        break;
    case 'c':
        if(optind > argc){
            break;
        }
        image.set_center(atoi(optarg)-1, atoi(argv[optind])-1);
        break;
    case 's':
        if(optind > argc){
            break;
        }
        image.set_point(0, atoi(optarg)-1, atoi(argv[optind])-1);
        break;
    case 'e':
        if(optind >= argc){
            break;
        }
        image.set_point(2, atoi(optarg)-1, atoi(argv[optind])-1);
        break;
    case 'R':
        image.set_radius(atoi(optarg));
        break;
    case 'b':
        image.set_bold(atoi(optarg));
        break;
    case 'B':
        image.set_color(0, atoi(optarg));
        for(int i = 1; i < 4; i++){
            if(optind >= argc){
                std::cerr << "Too few color arguments\n";
                exit(0);
            }
            if(argv[optind][0] != '0' && atoi(argv[optind]) == 0)
            {
                std::cerr << "Too few color arguments\n";
                exit(0);
            };
            image.set_color(i, atoi(argv[optind]));
            optind++;
        }
        break;
    case 'f':
        image.set_fill(1);
        break;
    case 'F':
        image.set_color_fill(0, atoi(optarg));
        for(int i = 1; i < 4; i++){
            if(optind >= argc){
                std::cerr << "Too few color arguments\n";
                exit(0);
            }
            if(argv[optind][0] != '0' && atoi(argv[optind]) == 0)

```

```

{
    std::cerr << "Too few color arguments\n";
    exit(0);
};
image.set_color_fill(i, atoi(argv[optind]));
optind++;
}
break;
case 'V':
    image.set_axis(0);
    break;
case 'Z':
    image.set_axis(1);
    break;
case 'S':
    name_out_file = optarg;
    break;
case '?':
    break;
default:
    abort();
}
option_index++;
}
switch (function_flag) {
    case 1:
        image.reflection();
        image.write_png_file(name_out_file);
        break;
    case 2:
        image.paint_pentogram();
        image.write_png_file(name_out_file);
        break;
    case 3:
        image.paint_rectangle();
        image.write_png_file(name_out_file);
        break;
    case 4:
        image.paint_hexagon();
        image.write_png_file(name_out_file);
        break;
    case 0:
        break;
    default:
        printf("Function doesn't choose\n");
}

return 0;
}

```

2. Файл class_images.h

```

#ifndef CW_IMAGE_H
#define CW_IMAGE_H
#include <png.h>

```

```

class images{

```

```

private:
    char header[8];
    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 point[6][2] = {{0,0},{0,0},{-1,-1},{0,0},{0,0},{0,0}};
    struct Png* image = new struct Png;
    int RGBA[4] = {-1,-1,-1,-1};
    int RGBA_Fill[4] = {-1,-1,-1,-1};
    int center[2] = {-1,-1};
    int bold = 0;
    int fill = 0;
    int R = 0;
    int Ax = -1;
public:
    ~images(){
        delete image;
    }
    void read_png_file(char *file_name);
    void write_png_file(char *file_name);
    void drawpixel(int x,int y);
    void drawpixel_Fill(int x,int y);
    void drawLine(int x1, int y1, int x2, int y2);
    void drawLine_Fill(int x1, int y1, int x2, int y2);
    void reflection();
    void paint_pentogram();
    void paint_rectangle();
    void paint_hexagon();

    int is_error();
    int is_error_R();
    int is_error_Color();
    int is_error_Color_Fill();
    struct Png* get_info(){
        return image;
    };

    void set_center(int x1, int y1){
        center[0] = x1;
        center[1] = y1;
    }
    void set_point(int N, int x1, int y1){
        point[N][0] = x1;
        point[N][1] = y1;
    }
    void set_axis(int a){
        Ax = a;
    }
    void set_radius(int a){
        R = a;
    }
    void set_color(int i, int c){

```

```

        RGBA[i] = c;
    }
    void set_color_fill(int i, int c){
        RGBA_Fill[i] = c;
    }
    void set_bold(int b){
        bold = b;
    }
    void set_fill(int f){
        fill = f;
    }
};

#endif //CW_IMAGE_H

```

3. Файл class_images.cpp

```

#include <iostream>
#include <cstdlib>
#include <cstdio>
#include <cstring>
#define PNG_DEBUG 3
#include <png.h>
#include "class_images.h"

void images::read_png_file(char *file_name) {
    int x,y;

    /* open file and test for it being a png */
    FILE *fp = fopen(file_name, "rb");
    if (!fp){
        std::cout << "File could not be opened\n";
        exit(0);
    }

    fread(header, 1, 8, fp);
    if (png_sig_cmp((png_const_bytep)header, 0, 8)){// проверяет
сигнатуру PNG
        std::cout << "File is not recognized as a PNG\n";
        exit(0);
    }

    /* initialize stuff */
    image->png_ptr = png_create_read_struct(PNG_LIBPNG_VER_STRING, NULL,
NULL, NULL);//выделяет память и инициализирует структуру для чтения PNG

    if (!image->png_ptr){
        std::cout << "png_create_read_struct failed\n";
        exit(0);
    }

    image->info_ptr = png_create_info_struct(image->png_ptr); //выделяет
память и инициализирует структуру инфо PNG
    if (!image->info_ptr){
        std::cout << "png_create_info_struct failed\n";
        exit(0);
    }
}

```

```

    if (setjmp(png_jmpbuf(image->png_ptr))) { // хз, какая-то проверка
        std::cout << "Error during init_io\n";
        exit(0);
    }

    png_init_io(image->png_ptr, fp); //инициализирует ввод для PNG файла
    png_set_sig_bytes(image->png_ptr, 8); // считывает 8 байт сигнатуры

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

    image->width = png_get_image_width(image->png_ptr, image->info_ptr);
    image->height = png_get_image_height(image->png_ptr, image-
>info_ptr);
    image->color_type = png_get_color_type(image->png_ptr, image-
>info_ptr);
    image->bit_depth = png_get_bit_depth(image->png_ptr, image-
>info_ptr);

    if (png_get_color_type(image->png_ptr, image->info_ptr) !=
PNG_COLOR_TYPE_RGBA) {
        std::cout << "Wrong color type, color type of input file must be
RGBA";
        exit(0);
    }

    image->number_of_passes = png_set_interlace_handling(image->png_ptr);
    png_read_update_info(image->png_ptr, image->info_ptr);

    /* read file */
    if (setjmp(png_jmpbuf(image->png_ptr))) {
        std::cout << "Error during read_image\n";
        exit(0);
    }

    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));

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

    fclose(fp);
}
void images::write_png_file(char *file_name) {
    int x, y;
    /* create file */
    FILE *fp = fopen(file_name, "wb");
    if (!fp) {
        std::cout << "Output file could not be opened\n";
        exit(0);
    }

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

```



```

    if (!image->png_ptr){
        std::cout << "png_create_write_struct failed\n";
        exit(0);
    }

    image->info_ptr = png_create_info_struct(image->png_ptr);
    if (!image->info_ptr){
        std::cout << "png_create_write_struct failed\n";
        exit(0);
    }

    if (setjmp(png_jmpbuf(image->png_ptr))){
        std::cout << "error during init_io\n";
        exit(0);
    }

    png_init_io(image->png_ptr, fp);

    /* write header */
    if (setjmp(png_jmpbuf(image->png_ptr))){
        std::cout << "error during writing header\n";
        exit(0);
    }

    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);

    /* write bytes */
    if (setjmp(png_jmpbuf(image->png_ptr))){
        std::cout << "error during writing bytes\n";
        exit(0);
    }

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

    /* end write */
    if (setjmp(png_jmpbuf(image->png_ptr))){
        std::cout << "error during end of write\n";
        exit(0);
    }

    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);
}

```

4. Файл reflection.cpp

```
#include <iostream>
#include <cstdlib>
#include <cstdio>
#include "class_images.h"
#include <png.h>

void images::reflection() {
    if (is_error()){
        std::cout<<"Wrong coordinates\n";
        exit(0);
    }
    int left_h = point[0][1], righth_h = point[2][1], left_w = point[0]
[0], righth_w = point[2][0];

    if(Ax == 0){
        for (int y = left_h; y < righth_h; y++) {
            png_byte *row = image->row_pointers[y];
            for (int x = left_w; x < righth_w/2; x++) {
                png_byte *ptr_beg = &(row[x * 4]);
                png_byte *ptr_eng = &(row[(righth_w - x - 1) * 4]);
                for(int i = 0;i < 4;i++){
                    png_byte ptr_swap;
                    ptr_swap = ptr_beg[i];
                    ptr_beg[i] = ptr_eng[i];
                    ptr_eng[i] = ptr_swap;
                }
            }
        }
    }
    else if (Ax == 1){
        for (int y = left_h; y < righth_h/2; y++) {
            png_byte *row_beg = image->row_pointers[y];
            png_byte *row_end = image->row_pointers[righth_h - y - 1];
            for (int x = left_w; x < righth_w; x++) {
                png_byte *ptr_beg = &(row_beg[x * 4]);
                png_byte *ptr_eng = &(row_end[x * 4]);
                for (int i = 0; i < 4; i++) {
                    png_byte ptr_swap;
                    ptr_swap = ptr_beg[i];
                    ptr_beg[i] = ptr_eng[i];
                    ptr_eng[i] = ptr_swap;
                }
            }
        }
    }
    else{
        std::cout << "Don't choose axis\n";
        exit(0);
    }
}
```

5. Файл paint_pentogram.cpp

```
#include <iostream>
#include <cstdlib>
#include <png.h>
#include "class_images.h"
```

```

void images::drawpixel(int x,int y){
    png_byte *row = image->row_pointers[y];
    for(int i = 0;i < 4;i++){
        row[x*4+i] = RGBA[i];
    }
}

void images::drawpixel_Fill(int x,int y){
    png_byte *row = image->row_pointers[y];
    for(int i = 0;i < 4;i++){
        row[x*4+i] = RGBA_Fill[i];
    }
}

void images::drawLine(int x1, int y1, int x2, int y2) {
    const int deltaX = abs(x2 - x1);
    const int deltaY = abs(y2 - y1);
    const int signX = x1 < x2 ? 1 : -1;
    const int signY = y1 < y2 ? 1 : -1;
    int error = deltaX - deltaY;
    drawpixel(x2,y2);
    while(x1 != x2 || y1 != y2)
    {
        drawpixel(x1,y1);
        int error2 = error * 2;
        if(error2 > -deltaY)
        {
            error -= deltaY;
            x1 += signX;
        }
        if(error2 < deltaX)
        {
            error += deltaX;
            y1 += signY;
        }
    }
}

void images::drawLine_Fill(int x1, int y1, int x2, int y2) {
    const int deltaX = abs(x2 - x1);
    const int deltaY = abs(y2 - y1);
    const int signX = x1 < x2 ? 1 : -1;
    const int signY = y1 < y2 ? 1 : -1;
    int error = deltaX - deltaY;
    drawpixel_Fill(x2,y2);
    while(x1 != x2 || y1 != y2)
    {
        drawpixel_Fill(x1,y1);
        int error2 = error * 2;
        if(error2 > -deltaY)
        {
            error -= deltaY;
            x1 += signX;
        }
        if(error2 < deltaX)

```

```

        {
            error += deltaX;
            y1 += signY;
        }
    }
}

void images::paint_pentogram(){

    if(center[0] < 0 || center[1] < 0 || R <= 0){
        if((point[2][0] - point[0][0]) != (point[2][1] - point[0][1])){
            std::cerr<<"Coordinates state not square\n";
            exit(0);
        }
        center[0] = point[0][0] + (point[2][0] - point[0][0])/2;
        center[1] = point[0][1] + (point[2][1] - point[0][1])/2;
        R = (point[2][0] - point[0][0])/2;
    }

    point[0][0] = center[0];
    point[0][1] = center[1];

    if(bold > R/2){
        std::cout<<"Too bold board\n";
        exit(0);
    }
    if (is_error_R()){
        std::cout<<"Wrong coordinates\n";
        exit(0);
    }
    if (is_error_Color()){
        std::cout<<"Wrong color\n";
        exit(0);
    }
}

// для рисования круга используем алгоритм Брезенхема
int X1 = center[0], Y1 = center[1];
int x = 0;
int y = R;
int delta = 1 - 2 * R;
int error = 0;

//Точки для пентограммы
point[0][0] = X1;
point[0][1] = Y1-R+bold;

//Рисование круга
while (y >= x){
    if(x == R/2){
        point[2][0] = X1-(R)+bold;
        point[2][1] = Y1-(R-y);
        point[3][0] = X1+(R)-bold;
        point[3][1] = Y1-(R-y);
    }
    if(x == R/2){
        point[1][0] = X1+x;

```

```

        point[1][1] = Y1+y-bold;

        point[4][0] = X1-x;
        point[4][1] = Y1+y-bold;
    }
    for(int i = 0; i < bold;i++){
        drawpixel(X1 + x, Y1 + y-i);
        drawpixel(X1 + x, Y1 - y+i);
        drawpixel(X1 - x, Y1 + y-i);
        drawpixel(X1 - x, Y1 - y+i);
        drawpixel(X1 + y-i, Y1 + x);
        drawpixel(X1 + y-i, Y1 - x);
        drawpixel(X1 - y+i, Y1 + x);
        drawpixel(X1 - y+i, Y1 - x);
    }
    error = 2 * (delta + y) - 1;
    if ((delta < 0) && (error <= 0)){
        delta += 2 * ++x + 1;
        continue;
    }
    if ((delta > 0) && (error > 0)){
        delta -= 2 * --y + 1;
        continue;
    }
    delta += 2 * (++x - --y);
}

//Рисование пентогаммы в круге
for(int i = 0; i < bold;i++){
    drawLine(point[0][0]+i,point[0][1],point[1][0]+i,point[1][1]);
    drawLine(point[1][0],point[1][1]+i,point[2][0],point[2][1]+i);
    drawLine(point[2][0],point[2][1]-i,point[3][0],point[3][1]-i);
    drawLine(point[3][0],point[3][1]+i,point[4][0],point[4][1]+i);
    drawLine(point[4][0]-i,point[4][1],point[0][0]-i,point[0][1]);
}
}

```

6. Файл paint_rectangle.cpp

```

#include <iostream>
#include <png.h>
#include "class_images.h"

void images::paint_rectangle(){
    point[1][0] = point[2][0];
    point[1][1] = point[0][1];
    point[3][0] = point[0][0];
    point[3][1] = point[2][1];

    if (is_error()){
        std::cout<<"Wrong coordinates\n";
        exit(0);
    }

    if((bold > (point[3][1]-point[0][1]) || bold > (point[1][0]-point[3]

```

```

[0])) && bold != 0){
    std::cerr << "Too bold board\n";
    exit(0);
}

if(fill == 1){
    if (is_error_Color_Fill()){
        std::cout<<"Wrong color fill\n";
        exit(0);
    }
    int paint;
    if((point[3][1]-point[0][1]) >= (point[3][0]-point[1][0])){
        paint = (point[3][1]-point[0][1])/2+1;
    }else{
        paint = (point[3][0]-point[1][0])/2+1;
    }

    for(int i = 0; i < paint; i++){
        drawLine_Fill(point[0][0],point[0][1]+i,point[1][0],point[1]
[1]+i);
        drawLine_Fill(point[1][0]-i,point[1][1],point[2][0]-
i,point[2][1]);
        drawLine_Fill(point[2][0],point[2][1]-i,point[3][0],point[3]
[1]-i);
        drawLine_Fill(point[3][0]+i,point[3][1],point[0]
[0]+i,point[0][1]);
    }
}

if (is_error_Color()){
    std::cout<<"Wrong color\n";
    exit(0);
}

for(int i = 0; i < bold; i++){
    drawLine(point[0][0],point[0][1]+i,point[1][0],point[1][1]+i);
    drawLine(point[1][0]-i,point[1][1],point[2][0]-i,point[2][1]);
    drawLine(point[2][0],point[2][1]-i,point[3][0],point[3][1]-i);
    drawLine(point[3][0]+i,point[3][1],point[0][0]+i,point[0][1]);
}
}

```

7. Файл paint_hexagon.cpp

```

#include <iostream>
#include "class_images.h"

void images::paint_hexagon(){
    if(center[0] < 0 || center[1] < 0 || R <= 0){
        if((point[2][0] - point[0][0]) != (point[2][1] - point[0][1])){
            std::cerr<<"Coordinates state not square\n";
            exit(0);
        }
        center[0] = point[0][0] + (point[2][0] - point[0][0])/2;
    }
}

```

```

        center[1] = point[0][1] + (point[2][1] - point[0][1])/2;
        R = (point[2][0] - point[0][0])/2;
    }

    point[0][0] = center[0];
    point[0][1] = center[1];

    R = R % 2 == 0 ? R : R-1;

    if(bold < 1 ){
        std::cerr << "Enter bold\n";
        exit(0);
    }
    if(R < 1){
        std::cerr << "Enter radius\n";
        exit(0);
    }

    if(bold > R){
        std::cerr << "Too bold board\n";
        exit(0);
    }

    int X1 = point[0][0], Y1 = point[0][1];
    int x = 0;
    int y = R;
    int delta = 1 - 2 * R;
    int error = 0;
    point[1][0] = X1+R;
    point[1][1] = Y1;
    int X2 = point[1][0], Y2 = point[1][1];
    //Точки для шестиугольника
    point[0][0] = X1+R;
    point[0][1] = Y1;
    point[1][0] = X1-R;
    point[1][1] = Y1;

    // Чтобы построить правильный шестиугольник, воспользуемся алгоритмом
    Евклида, использующего
    // точки пересечения окружностей, точки пересечения найдем с помощью
    алгоритма Брезенхема.

    while (y >= x){
        if((X1 + x) == (X2 - x) && (Y1 + y) == (Y2 + y)){
            point[2][0] = X1 + x;
            point[2][1] = Y1 + y;
            point[3][0] = X1 + x;
            point[3][1] = Y1 - y;
            point[4][0] = X1 - x;
            point[4][1] = Y1 + y;
            point[5][0] = X1 - x;
            point[5][1] = Y1 - y;
            break;
        }
        error = 2 * (delta + y) - 1;
        if ((delta < 0) && (error <= 0)){
            delta += 2 * ++x + 1;
            continue;
        }
    }

```

```

    }
    if ((delta > 0) && (error > 0)){
        delta -= 2 * --y + 1;
        continue;
    }
    delta += 2 * (++x - --y);
}

if(point[3][1] < 0 || point[2][1] > image->height-1 || point[0][0] >
image->width-1 || point[1][0] < 0){
    std::cout<<"Wrong coordinates\n";
    exit(0);
}

if(fill == 1){
    if(rgba_Fill[0] == -1 || rgba_Fill[0] == -1 || rgba_Fill[0] == -1
|| rgba_Fill[0] == -1){
        std::cout<<"Enter correct color\n";
        exit(0);
    }
    if (is_error_Color_Fill()){
        std::cout<<"Wrong color\n";
        exit(0);
    }

    int paint = R;
    for(int i = 0; i < paint; i++){
        drawLine_Fill(point[0][0]-i, point[0][1], point[3][0]-
i, point[3][1]);
        drawLine_Fill(point[3][0], point[3][1]+i, point[5][0], point[5]
[1]+i);
        drawLine_Fill(point[5][0]+i, point[5][1], point[1]
[0]+i, point[1][1]);
        drawLine_Fill(point[1][0]+i, point[1][1], point[4]
[0]+i, point[4][1]);
        drawLine_Fill(point[4][0], point[4][1]-i, point[2][0], point[2]
[1]-i);
        drawLine_Fill(point[2][0]-i, point[2][1], point[0][0]-
i, point[0][1]);
    }
}

if (is_error_Color()){
    std::cout<<"Wrong color\n";
    exit(0);
}

//Рисуем шестиугольник
for(int i = 0; i < bold; i++){
    drawLine(point[0][0]-i, point[0][1], point[3][0]-i, point[3][1]);
    drawLine(point[3][0], point[3][1]+i, point[5][0], point[5][1]+i);
    drawLine(point[5][0]+i, point[5][1], point[1][0]+i, point[1][1]);
    drawLine(point[1][0]+i, point[1][1], point[4][0]+i, point[4][1]);
    drawLine(point[4][0], point[4][1]-i, point[2][0], point[2][1]-i);
    drawLine(point[2][0]-i, point[2][1], point[0][0]-i, point[0][1]);
}
};

```


8. Файл is_error.cpp

```
#include "class_images.h"

int images::is_error(){
    if(point[0][0] > image->width-1 || point[0][1] > image->height-1 ||
    point[2][0] > image->width-1 || point[2][1] > image->height-1 ||
    point[0][0] < 0 || point[0][1] < 0 || point[2][0] < 0 || point[2][1]
< 0 ||
    point[0][0] > point[2][0] || point[0][1] > point[2][1]
    ){
        return 1;
    }
    return 0;
};

int images::is_error_R(){
    if(point[0][0] + R > image->width-1 || point[0][1] + R > image-
>height-1 ||
    point[0][0] - R < 0 || point[0][1] - R < 0 ||
    point[0][0] < 0 || point[0][1] < 0 || R <= 0){
        return 1;
    }
    return 0;
}

int images::is_error_Color(){
    if(RGBA[0] > 255 || RGBA[1] > 255 || RGBA[2] > 255 || RGBA[3] > 255
|| RGBA[0] < 0 || RGBA[1] < 0 || RGBA[2] < 0 || RGBA[3] < 0){
        return 1;
    }
    return 0;
}

int images::is_error_Color_Fill(){
    if(RGBA_Fill[0] > 255 || RGBA_Fill[1] > 255 || RGBA_Fill[2] > 255 ||
RGBA_Fill[3] > 255 || RGBA_Fill[0] < 0 ||
    RGBA_Fill[1] < 0 || RGBA_Fill[2] < 0 || RGBA_Fill[3] < 0){
        return 1;
    }
    return 0;
}
}
```

9. Makefile.

```
all: main.o class_images.o reflection.o paint_pentogram.o
paint_rectangle.o paint_hexagon.o is_error.o
g++ main.o class_images.o reflection.o paint_pentogram.o
paint_rectangle.o paint_hexagon.o is_error.o -lpng
```

```
main.o: main.cpp class_images.h
g++ -c main.cpp
```

```
class_images.o: class_images.cpp class_images.h
g++ -c class_images.cpp
```

```
reflection.o: reflection.cpp class_images.h
g++ -c reflection.cpp

paint_pentogram.o: paint_pentogram.cpp class_images.h
g++ -c paint_pentogram.cpp

paint_rectangle.o: paint_rectangle.cpp class_images.h
g++ -c paint_rectangle.cpp

paint_hexagon.o: paint_hexagon.cpp class_images.h
g++ -c paint_hexagon.cpp

is_error.o: is_error.cpp class_images.h
g++ -c is_error.cpp

clean:
    rm *.o a.out
```