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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка PNG файла

Студент гр. 2300 _____ Локосов Д.Д.

Преподаватель _____ Гаврилов А.В.

Санкт-Петербург

2023

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Локосов Д.Д.

Группа 2300

Тема работы: Обработка PNG файла

Исходные данные:

Вариант 16

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.

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:
    - Цветом, который требуется заменить
    - Цветом, на который требуется заменить

3. Сделать рамку в виде узора. Рамка определяется:
    - Узором (должно быть несколько на выбор)
    - 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

Содержание пояснительной записки:
разделы «Аннотация», «Содержание», «Введение», «Ход работы», «Пример работы программы», «Заключение», «Список использованных источников»

Предполагаемый объем пояснительной записки:
Не менее 10 страниц.

Дата выдачи задания: 22.03.2023

Дата сдачи реферата: 18.05.2023

Дата защиты реферата: 20.05.2023

Студент _____ Локосов Д.Д.

Преподаватель _____ Гаврилов А.В.

АННОТАЦИЯ

Курсовая работа представляет собой реализацию программы обработки PNG файла, данного пользователем, на языке программирования C.

Для ввода/вывода изображения была использована библиотека *libpng*.

Для взаимодействия с программой был сделан CLI.

В результате написана программа, реализованная в виде утилиты. При неправильном вводе печатается сообщение об ошибки. Также может быть выведена справка о программе с описанием синтаксиса утилиты и возможностях. Программа выполняет считывание изображения, его обработку и сохранение в файле, указанном пользователем.

Для запуска программы требуется зайти в корневую папку с проектом, ввести в консоль *make*. Далее можно вывести подсказку с помощью *./cw_png -h*.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной работы является освоение подходов к работе с PNG файлами в языке C и последующая реализация программы для их обработки. Для достижения поставленной цели требуется решить следующие задачи:

1. Реализовать способ хранения изображения с помощью структуры.
2. Реализовать ввод/вывод изображения.
3. Реализовать базовые функции для обработки изображения.
4. Реализовать задания из курсовой работы на обработку изображения в соответствии с условием варианта в виде функций.
5. Реализовать интерфейс при помощи *CLI* и библиотеки *getopt*
6. Сгруппировать написанные функции по файлам в соответствии с выполняемыми ими задачами.
7. Написать *Makefile* для сборки проекта.

2. ХОД ВЫПОЛНЕНИЯ РАБОТЫ

2.1. Создание структур для хранения изображения

Для хранения данных об изображении были созданы структуры *Image,* а для хранения информации о конкретном пикселе – *Pixel*.

Структура Image включает в себя следующие основные поля:

- *int width, height* – размер изображения
- *png_bytepp rowPointers* – массив указателей на указатели на пиксели

Структура *Pixel* включает в себя следующие поля:

- *png_byte red, green, blue, alpha* – компоненты цвета

Также для быстрой инициализации структуры Image была написана функция *Image *createImg(),* которая возвращают структуры с обнуленными переменными. И функция *void clearImage(Image *img)* для очистки полей структуры.

Для объявления структур и функций был создан отдельный заголовочный файл *png_structs.h,* а для определения функций – *png_structs.c*.

2.2. Ввод/вывод изображения

Для ввода изображения используется функция *void readPngFile(const char *fileName, Image *img)*, принимающая на вход имя читаемого файла и указатель на структуру, в которую будет записана информация об изображении. Функция открывает файл, считывает его заголовок. После чего благодаря встроенным в *libpng* функциям происходит считывание изображения, заполнение полей структуры и обработка ошибок.

Вывод изображения производится с помощью функции *void writePngFile(const char *fileName, Image *img),* принимающей на вход имя записываемого файла и указатель на структуру с изображением. Далее применяя практически аналогичные функции в сравнении с вводом изображения, файл заполняется данными об изображении.

Также написаны вспомогательные функции *Pixel getPixel(int x, int y, Image *img)* и *void putPixel(int x, int y, Pixel pix, Image *img)*, которые по координатам и указателю на структуру с изображением способны получить либо вставить пиксель соответственно.

Для данных функций были созданы отдельные файлы *png_io.c* и *png_io.h*.

2.3. Вспомогательные функции обработки изображения

В целях упрощения дальнейшей обработки изображения были реализованы некоторые вспомогательные функции:

- *int pixCmp(Pixel first, Pixel second, int CPP)* – покомпонентное сравнение двух пикселей
- *void swap(void *first, void *second, size_t size)* – универсальная функция замены двух переменных с известным размером
- *double fractalKali(double x, double y, double cx, double cy, int width, int height)* и аналогичные по сигнатуре *fractalJulia* и *fractalBio* – функции фракталов, которые по координате, «свободным» значениям и размерам фрактала возвращают 1, если данная точка принадлежит фракталу, и 0 в противном случае.
- *int cmpGreat(double first, double second)* и аналогичная *cmpLess* – компараторы. Первая проверяет условие first > second, вторая first < second.
- *int isPrime(int n)* – проверяет число n на простоту.
- *int XOR(int a, int b)* и аналогичные по сигнатуре *AND* и *OR* – возвращают результат выполнения над двумя числами «исключающего или», побитового «и» и «или» соответственно.

Для данных функций были созданы отдельные файлы *png_process_support.c* и *png_process_support.h*.

2.4. Обработка изображения

Пользователю на выбор даётся 4 действия над изображением: копирование области, замена пикселей одного цвета на другой, рисование рамки, обводка всех залитых прямоугольников.

Копирование области:

Для данной задачи реализована функция *void copyArea(int x1Src, int y1Src, int x2Src, int y2Src, int xDst, int yDst, Image *img)*. Она принимает на вход координаты левой верхней и правой нижней точек области-источника и координаты левой верхней точки области-назначения. Перед самим процессом копирования производится подготовка входных данных: замена координат, если они были даны в неправильном порядке; нормализация координат, если они выходят за границы изображения; определяются размеры копируемой области. Далее создаётся двумерный массив пикселей, который заполняется копируемой областью. После чего данные с массива переносятся в область-назначение.

Замена цвета:

Реализована функция *void changeColor(Pixel src, Pixel dst, Image *img)*. Она принимает заменяемый и конечный цвет. Далее функция «пробегает» по каждому пикселю, сравнивает его с данным и в случае совпадения вставляет на его место пиксель конечного цвета.

Обводка прямоугольников:

Реализована функция *void circleFilledRect(Pixel rectCol, Pixel strokeCol, int width, Image *img)*, принимающая цвет искомых прямоугольников, цвет и ширину обводки. Функция пробегает по каждому пикселю и если он равен искомому (и при этом он не находится в найденных прямоугольниках, что проверяется функцией *int isDotInRects(int x, int y, int **cords, int cordLen)*), то начинается «расширение» области до залитого прямоугольника, таким образом получается координата нижнего правого угла прямоугольника, а левый верхний был найден при первом совпадении цвета. Далее эта пара координат сохраняется в массиве. После проверки всего изображения по полученным координатам обводятся все прямоугольники с помощью функции *void drawStraightLine(int x1,*

*int y1, int x2, int y2, Pixel color, Image *img)*, которая позволяет рисовать вертикальные или горизонтальные линии.

Рисование рамки:

Реализована функции *void drawFrameFrac(double (*frac)(double x, double y, double cx, double cy, int width, int height), int (*inv)(double first, double second), double style, int form, Pixel color, int width, Image *img)* и *void drawFramePrime(int (*bitOp)(int a, int b), int (*inv)(double first, double second), double style, int form, Pixel color, int width, Image *img)*. Они отличаются лишь функцией задания узора (одна рисует фракталы, другая – узоры простых чисел). Они принимают функцию-узор, компаратор (для реализации инвертирования узора), свободный параметр, влияющий на внешний вид узора, цвет и ширину рамки, а также тип относительного размера узора (рамка может быть образована либо обрезкой узора наложенного на всё изображение, либо быть собранной из нескольких блоков-узоров). Далее изображение расширяется на ширину рамки * 2 с каждой стороны при помощи функции *void expandImg(Image *img, int width)*. Определяются вспомогательные переменные в виде размера узора, сбалансированного размера (чтобы в случае блочного строения блоки не обрезались). После чего начинается собственно рисование нижней/верхней рамки и левой/правой (их рисовку удобно разместить в одном цикле, поскольку они должны быть симметричными).

Для данных функций были созданы отдельные файлы *png_process.c* и *png_process.h.*

2.5. Взаимодействие с пользователем

Для взаимодействия с пользователем был создан *CLI* с помощью библиотеки *getopt.* Для этого создаётся структура *Configs* и создаётся экземпляр с значениями по умолчанию при помощи функции *Configs createConfig()*. Перед этим было проверено, что пользователь ввёл достаточно количество аргументов для работы утилиты, в противном случае вызывается справка. Далее изображение обрабатывается и проверяется на совпадение типа. После чего с

помощью функции *getopt_long()* и шаблонов ключей в виде строки коротких ключей *opts* и массива структур длинных ключей *longOpts* производится обход всех ключей, которые обрабатываются функцией *int processOpt(int opt, Configs *config)*, в случае неверного ввода значения вызывается ошибка и обход ключей прекращается. Функция заполняет соответствующие поля структуры конфигов. Если это ключ функции, то она возвращает 1, что говорит внешней функции main о необходимости вызова соответствующей функции обработки, после чего обход продолжается. Если ключ предполагает вызов памятки, то она печатается функцией *printHelp()*. Если ключ предполагает вызов информации о файле, то после пробега всех ключей, обработки изображения, будет выведена информация о конечном файле.

Для функций обработки ключей и вызова справки созданы отдельные файлы *cli.c* и *cli.h*.

2.6. Написание Makefile

Основной особенностью написания *Makefile* в данной работе служит использование сторонней библиотеки *libpng*, что вынуждает использовать при компиляции и линковке файлов дополнительный флаг *-lpng*. Также из-за использования математических функций был добавлен флаг *-lm*.

Таким образом, переменная с флагами приняла вид *CFLAGS=-c -Wall -lpng -lm*

А переменная с компилятором *CC=gcc*

Далее были проведены стандартные действия по выставлению целей, зависимостей и действий. Также была реализована автоматическая очистка объектных файлов.

ЗАКЛЮЧЕНИЕ

Были изучены основные функции библиотеки libpng для считывания/сохранения PNG файла

Была написана CLI утилита, принимающая обрабатываемый файл, несколько ключей-конфигураторов, ключи функций и выходной файл в конце (опционально). Далее файл считывается, обрабатывается и сохраняется. Таким образом, пользователь может скопировать область на изображении, заменить цвет, нарисовать рамку, обвести залитые прямоугольники. Притом за один вызов утилиты можно выполнить несколько обработок.

Для сборки проекта курсовой работы был написан *Makefile*.

Примеры работы программы см. в приложении А.

Разработанный программный код см. в приложении Б.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Керниган Б., Ритчи Д. Язык программирования Си. СПб.: "Невский Диалект", 2001. 352 с.

2. Libpng manual: [Электронный ресурс]. URL: https://libpng.org/

3. Cplusplus: [Электронный ресурс]. URL: https://cplusplus.com/

4. C-cpp: [Электронный ресурс]. URL: https://www.c-cpp.ru/

ПРИЛОЖЕНИЕ А
ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

Пример 1 (см. рис. 1):

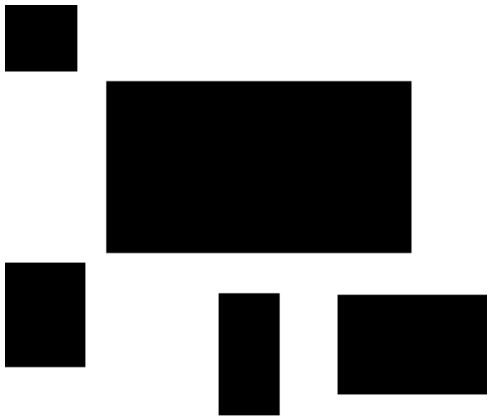
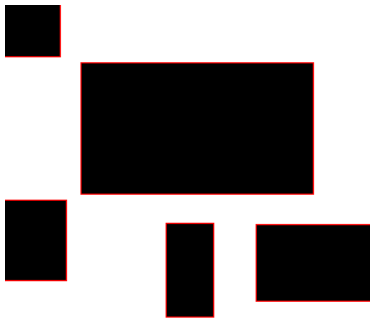


Рисунок 1 – несколько черных залитых прямоугольников

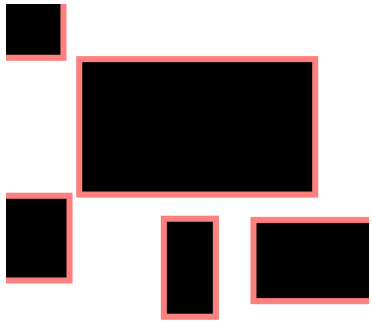
1. Обводка прямоугольников красной линией

```
./cw_png rect.png -C 255.0.0 -l res.png
```



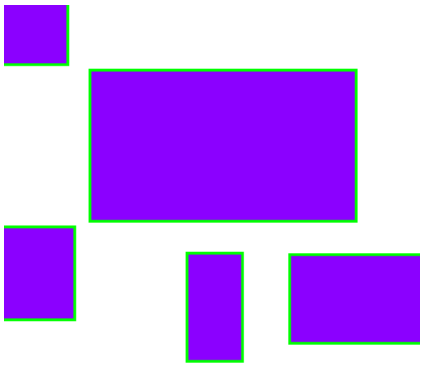
2. Обводка красным полупрозрачным цветом шириной 5

```
./cw_png rect.png -C 255.0.0.127 -w 5 -l res.png
```



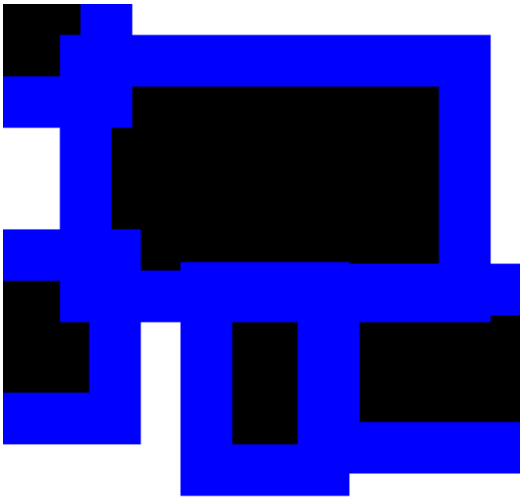
3. Обводка прямоугольника зеленым цветом ширины 2 и замена черных пикселей на фиолетовые

```
./cw_png rect.png -C 0.255.0 -w 2 -l -c 0.0.0 -C 139.0.255 -r res.png
```



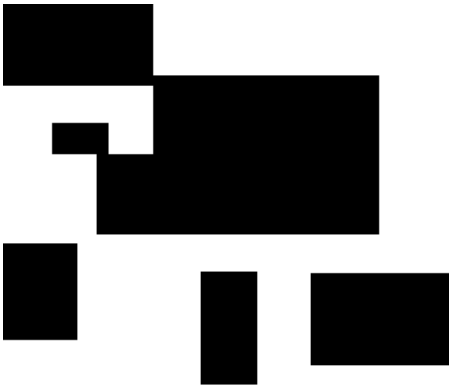
4. Обводка прямоугольников синей линией ширины 30

```
./cw_png rect.png -C 0.0.255 -w 30 -l res.png
```



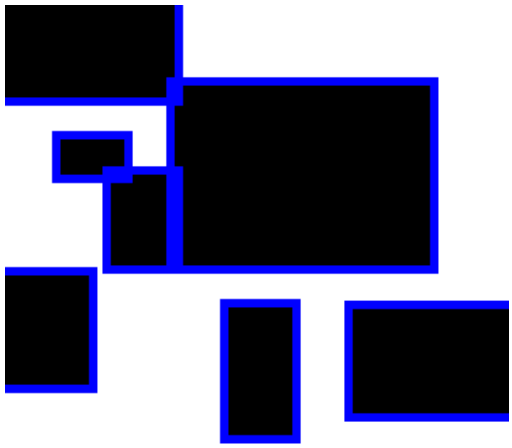
5. Копирование области из (100; 100) – (200; 200) в (0; 0)

```
./cw_png rect.png --first 100.100 --second 200.200 -d res.png
```



6. Предыдущий шаг + обводка прямоугольников синей линии ширины 5

```
./cw_png rect.png -f 100.100 -s 200.200 -d -C 0.0.255 -w 5 -l res.png
```



7. Вывод памятки

```
user@HONORbook:/mnt/c/Users/lokos/CLionProjects/course_work_2/cmake-build-debug$ ./cw_png

НАИМЕНОВАНИЕ
       cw_png - утилита для обработки PNG изображений
СИНТАКСИС
       cw_png [имя исходного файла] [ключи-конфигураторы] [ключ функции 1] [...] [опционально: имя выходного файла]
ОПИСАНИЕ
ФУНКЦИИ
       -d/--duplicate - копирование заданной области по координатам левого верхнего (-f), правого нижнего (-s) углов области-источника, а также левого верхнего (-t) угла области-п
олучателя.
       -r/--repaint - замена всех пикселей изображения заданного цвета (-c) на другой заданный цвет (-C).
       -l/--locate-rect - поиск всех залитых прямоугольников заданного цвета (-c) и их обводка линиями заданного цвета (-C) определенной ширины (-w).
       -b/--bezel - рисование рамки ширины (-w) (при маленьких значениях рамка может пропасть) и цвета (-C) по краям изображения. Вид рамки можно менять несколькими настройками:
          -Тип используемого фрактала для получения узора (-T): значение от 0 до 5. По умолчанию 0.
               -0-2 - фрактальные узоры. 1 задаёт множество Жюлиа (хорошие узоры при стиле около 5). 2 задаёт биоморфов.
               -2-5 - узоры простых чисел. 3 - узор микросхемы, 4 - подобие фрактальных треугольников, 5 - нечто среднее.
          -Инвертирование узора (-I).
          -Параметр стиля узора (-S): Допустимы вещественные числа. Данный параметр качественно меняет узор. Каждый узор меняется по-своему.
          -Форма заливки узора (-F). Без флага рамка строится из блоков-узоров. С флагом рамка менее однородная, поскольку получена обрезкой одного узора.
КОНФИГУРАЦИИ
       -f/--first <x.y> - координаты первой точки, требуемой функцией. По умолчанию 0.0.
       -s/--second <x.y> - координаты второй точки, требуемой функцией. По умолчанию 0.0.
       -t/--third <x.y> - координаты третьей точки, требуемой функцией. По умолчанию 0.0.
       -w/--width <value> - задаёт толщину отрезка/рамки. По умолчанию 1.
       -c/--src-col <R.G.B/R.G.B.A> - задаёт искомый цвет. По умолчанию 0.0.0.255.
       -c/--dst-col <R.G.B/R.G.B.A> - задаёт конечный цвет. По умолчанию 0.0.0.255.
       -S/--style <value> - рекомендуется значения от 0 до 10, допустимы вещественные. Хорошие узоры получаются в диапазоне 7 - 9. По умолчанию 8.
       -T/--type <0-5> - меняет тип узора.
       -I/--invert - инвертирует узор.
```

8. Вывод информации о файле

```
user@HONORbook:/mnt/c/Users/lokos/CLionProjects/course_work_2/cmake-build-debug$ ./cw_png rect.png -i
WIDTH: 300
HEIGHT: 300
COLOR TYPE: 6
BIT DEPTH: 8
COLOR CHANNELS: 4
```

Пример 2 (см. рис. 2):

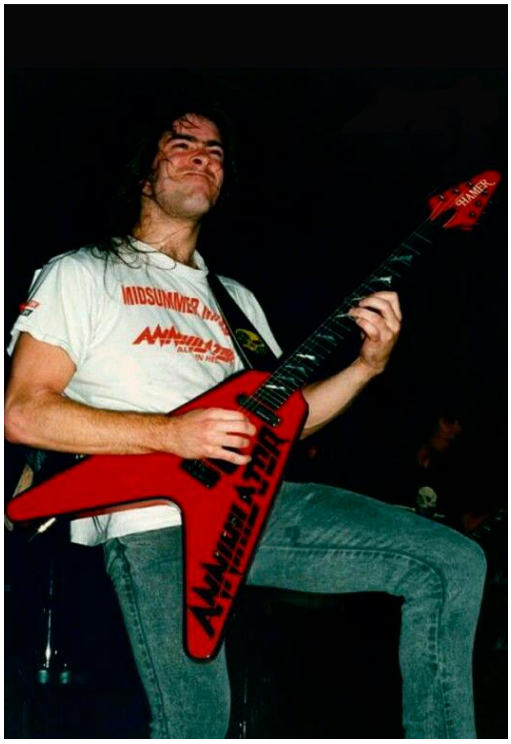
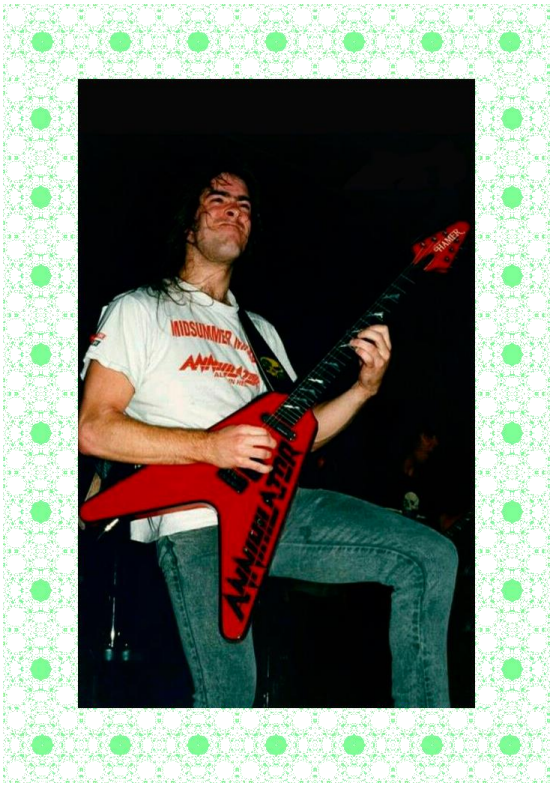


Рисунок 2 – канадский гитарист-виртуоз

1. Рисование рамки по умолчанию

```
./cw_png test.png --width 100 -C 125.255.150 -b res.png
```



2. Рисование той же рамки с измененным стилем

```
./cw_png test.png --width 100 -C 50.100.255 -S 9 -b res.png
```

3. Рисование предыдущей рамки во «внутрь»

```
./cw_png test.png --width -100 -C 50.100.255 -S 9 -b res.png
```

4. Рисование инвертированного узора множества Жюлиа

```
./cw_png test.png --width 100 -C 50.100.255 -T 1 -S 3.9 -I -b res.png
```

5. Рисование узора-биоморф

```
./cw_png test.png -w 100 -C 50.100.255 -T 2 -S 20 -b res.png
```

6. Рисование того же узора с другим стилем

```
./cw_png test.png -w 100 -C 50.100.255 -T 2 -S 5 -b res.png
```

7. Рисование узора простых чисел – микросхема с обрезкой

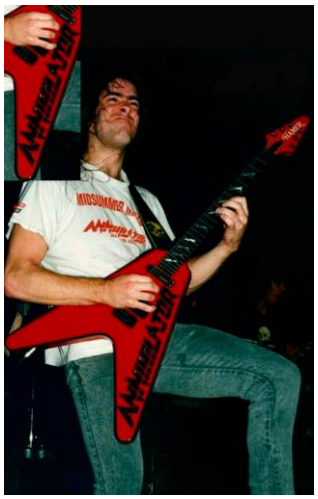
```
./cw_png test.png -w 100 -C 50.100.255 -T 3 -S 7 -F -b res.png
```

8. Рисование узора по умолчанию с обрезкой

```
./cw_png test.png -w 50 -C 50.100.255 -F -b res.png
```

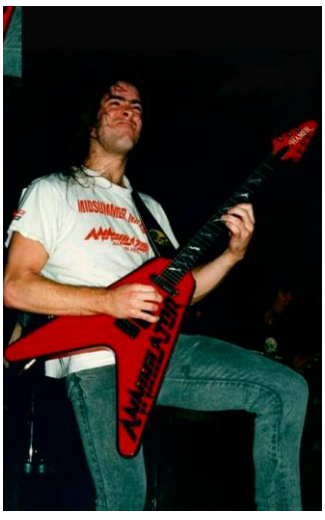
9. Копирование части области

```
./cw_png test.png -f 170.450 -s 300.750 -t 0.0 -d res.png
```



10. Копирование той же части «за пределы изображения»

```
./cw_png test.png -f 170.450 -s 300.750 -t -100.-100 -d res.png
```



ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

```
#include <getopt.h>
#include "png_io.h"
#include "error.h"
#include "png_structs.h"
#include "png_process.h"
#include "cli.h"

int main(int argc, char **argv)
{
    if (argc < 3)
    {
       printHelp();
       return 1;
    }
    char *input = argv[1];

    Configs config = createConfig();
    const char *opts = "f:s:t:w:c:C:T:S:o:FIrlbdih";
    const struct option longOpts[] = {
          {"first",       required_argument, NULL, 'f'},
          {"second",      required_argument, NULL, 's'},
          {"third",       required_argument, NULL, 't'},
          {"width",       required_argument, NULL, 'w'},
          {"src-col",     required_argument, NULL, 'c'},
          {"dst-col",     required_argument, NULL, 'C'},
          {"style",       required_argument, NULL, 'S'},
          {"type",        required_argument, NULL, 'T'},
          {"invert",      no_argument,       NULL, 'I'},
          {"form",        no_argument,       NULL, 'F'},
          {"output",      required_argument, NULL, 'o'},
          {"help",        no_argument,       NULL, 'h'},
          {"info",        no_argument,       NULL, 'i'},
          {"duplicate",   no_argument,       NULL, 'd'},
          {"repaint",     no_argument,       NULL, 'r'},
          {"bezel",       no_argument,       NULL, 'b'},
          {"locate-rect", no_argument,       NULL, 'l'},
          {NULL, 0,                          NULL, 0}
    };
    int opt;
    int longIndex;
    int condition;

    Image *img = createImg();
    readPngFile(input, img);

    opt = getopt_long(argc, argv, opts, longOpts, &longIndex);
    while (opt != -1)
    {
```

```
        condition = processOpt(opt, &config);
        if (condition == 1)
        {
           if (config.cpyArea)
           {
              config.cpyArea = 0;
              copyArea(config.x1Src, config.y1Src, config.x2Src,
config.y2Src, config.xDst, config.yDst, img);
           }
           else if (config.repaint)
           {
              config.repaint = 0;
              changeColor(config.srcColor, config.dstColor, img);
           }
           else if (config.circleRects)
           {
              config.circleRects = 0;
              circleFilledRect(config.srcColor, config.dstColor,
config.width, img);
           }
           else if (config.drawFrame)
           {
              config.drawFrame = 0;
              if (config.frameType <= 2)
                 drawFrameFrac(config.frameType == 1 ? fractalJulia :
config.frameType == 2 ? fractalBio : fractalKali,
                       config.frameInvert ? cmpLess : cmpGreat,
config.frameStyle, config.frameForm,
                       config.dstColor, config.width, img);
              else if (config.frameType <= 5)
                 drawFramePrime(config.frameType == 3 ? XOR :
config.frameType == 4 ? AND : OR,
                             config.frameInvert ? cmpLess : cmpGreat,
config.frameStyle, config.frameForm,
                             config.dstColor, config.width, img);
              else
                 panic("This type of frame does not exist");
           }
        }
        else if (condition == -1)
           break;

        opt = getopt_long(argc, argv, opts, longOpts, &longIndex);
     }

     if (config.info)
        printInfo(img);

     argc -= optind;
     argv += optind;

     if (argc && !config.output && strstr(argv[argc - 1], ".png"))
        config.output = argv[argc - 1];
     else if (!config.output)
        config.output = input;

     writePngFile(config.output, img);
```

```
    clearImage(img);

    return 0;
}
```

Название файла: png_structs.h

```
#pragma once

#include <png.h>
#include <stdlib.h>

typedef struct Image {
      int width, height;
      png_byte colorType, bitDepth;

      png_structp pngPtr;
      png_infop infoPtr;
      int passesNumber;
      png_bytepp rowPointers;
} Image;

typedef struct Pixel {
      png_byte red, green, blue, alpha;
} Pixel;

/* возвращает указатель на структуру Image с обнуленными переменными */
Image *createImg();

/* безопасно очищает поля структуры */
void clearImage(Image *img);
```

Название файла: png_structs.c

```
#include "png_structs.h"

Image *createImg()
{
      Image *img = (Image *) malloc(sizeof(Image));

      img->width = 0;
      img->height = 0;
      img->passesNumber = 0;
      img->colorType = 0;
      img->bitDepth = 0;
      img->pngPtr = NULL;
      img->infoPtr = NULL;
      img->rowPointers = NULL;

      return img;
}

void clearImage(Image *img)
{
      png_destroy_read_struct(&img->pngPtr, &img->infoPtr, NULL);
      if (img->rowPointers)
      {
```

```
            for (int y = 0; y < img->height; y++)
                  png_free(img->pngPtr, img->rowPointers[y]);
            png_free(img->pngPtr, img->rowPointers);
      }

      free(img);
}
```

Название файла: error.h

```
#pragma once

#include "png_structs.h"

/* высылает сообщение соответствующей ошибки в поток ошибок */
void panic(const char *err);

/* высылает сообщение ошибки, очищает структуру изображения и завершает
выполнение программы */
void crash(const char *err, Image *img);
```

Название файла: error.c

```
#include "error.h"

void panic(const char *errMsg)
{
      fprintf(stderr, "%s\n", errMsg);
}

void crash(const char *errMsg, Image *img)
{
      panic(errMsg);
      clearImage(img);
      exit(EXIT_FAILURE);
}
```

Название файла: png_io.h

```
#pragma once

#include <png.h>
#include "png_structs.h"
#include "error.h"

/* принимает имя файла и указатель на изображение, после чего сохраняет
изображение в файл */
void writePngFile(const char *fileName, Image *img);

/* принимает имя файла и указатель на изображение, после чего заполняет
его из файла */
void readPngFile(const char *fileName, Image *img);

/* возвращает структуру Pixel соответствующей координаты */
Pixel getPixel(int x, int y, Image *img);

/* ставит пиксель в соответствующую координату изображения */
void putPixel(int x, int y, Pixel, Image *img);
```

```
/* печатает основную информацию о PNG файле */
void printInfo(Image *img);
```

Название файла: png_io.c

```
#include "png_io.h"

void readPngFile(const char *fileName, Image *img)
{
	char header[8];

	FILE *fp = fopen(fileName, "rb");
	if (!fp)
		crash("Error during opening file for reading", img);

	fread(header, 1, 8, fp);
	if (png_sig_cmp((png_const_bytep) header, 0, 8))
		crash("File is not a PNG", img);

	img->pngPtr = png_create_read_struct(PNG_LIBPNG_VER_STRING, NULL,
NULL, NULL);
	if (!img->pngPtr)
		crash("Creating read struct failed", img);

	img->infoPtr = png_create_info_struct(img->pngPtr);
	if (!img->infoPtr)
		crash("Creating info struct failed", img);

	if (setjmp(png_jmpbuf(img->pngPtr)))
		crash("An error occurred during init_io", img);

	png_init_io(img->pngPtr, fp);
	png_set_sig_bytes(img->pngPtr, 8);

	png_read_info(img->pngPtr, img->infoPtr);

	img->width = (int) png_get_image_width(img->pngPtr, img->infoPtr);
	img->height = (int) png_get_image_height(img->pngPtr,
img->infoPtr);
	img->colorType = png_get_color_type(img->pngPtr, img->infoPtr);
	img->bitDepth = png_get_bit_depth(img->pngPtr, img->infoPtr);

	img->passesNumber = png_set_interlace_handling(img->pngPtr);
	png_read_update_info(img->pngPtr, img->infoPtr);

	if (setjmp(png_jmpbuf(img->pngPtr)))
		crash("An error occurred during reading img", img);

	img->rowPointers = png_malloc(img->pngPtr, img->height *
sizeof(png_bytep));
	for (int i = 0; i < img->height; i++)
		img->rowPointers[i] = png_malloc(img->pngPtr,
png_get_rowbytes(img->pngPtr, img->infoPtr));

	png_read_image(img->pngPtr, img->rowPointers);

	fclose(fp);
```

```
}

void writePngFile(const char *fileName, Image *img)
{
      FILE *fp = fopen(fileName, "wb");
      if (!fp)
            crash("Error during opening file for writing", img);

      img->pngPtr = png_create_write_struct(PNG_LIBPNG_VER_STRING, NULL,
NULL, NULL);
      if (!img->pngPtr)
            crash("Creating write struct failed", img);

      img->infoPtr = png_create_info_struct(img->pngPtr);
      if (!img->infoPtr)
            crash("Creating info struct failed", img);

      if (setjmp(png_jmpbuf(img->pngPtr)))
            crash("An error occurred during init_io", img);

      png_init_io(img->pngPtr, fp);

      if (setjmp(png_jmpbuf(img->pngPtr)))
            crash("An error occurred during header writing", img);

      png_set_IHDR(img->pngPtr, img->infoPtr, img->width, img->height,
img->bitDepth,
                        img->colorType, PNG_INTERLACE_NONE,
PNG_COMPRESSION_TYPE_BASE, PNG_FILTER_TYPE_BASE);

      png_write_info(img->pngPtr, img->infoPtr);

      if (setjmp(png_jmpbuf(img->pngPtr)))
            crash("An error occurred during writing bytes", img);

      png_write_image(img->pngPtr, img->rowPointers);

      if (setjmp(png_jmpbuf(img->pngPtr)))
            crash("An error occurred during end of write", img);

      png_write_end(img->pngPtr, NULL);

      fclose(fp);
}

Pixel getPixel(int x, int y, Image *img)
{
      if (x < 0 || x >= img->width || y < 0 || y >= img->height)
            crash("Image overstepping error", img);

      int colorChannels = png_get_channels(img->pngPtr, img->infoPtr);

      png_bytep ptr = img->rowPointers[y] + x * colorChannels;

      Pixel pix = {ptr[0], ptr[1], ptr[2], colorChannels > 3 ? ptr[3] :
255};
```

```
	return pix;
}

void putPixel(int x, int y, Pixel pix, Image *img)
{
	if (x < 0 || x >= img->width || y < 0 || y >= img->height)
		crash("Image overstepping error", img);

	int colorChannels = png_get_channels(img->pngPtr, img->infoPtr);

	png_bytep ptr = img->rowPointers[y] + x * colorChannels;

	ptr[0] = pix.red;
	ptr[1] = pix.green;
	ptr[2] = pix.blue;
	if (colorChannels > 3)
		ptr[3] = pix.alpha;
}

void printInfo(Image *img)
{
	printf("WIDTH: %d\n", img->width);
	printf("HEIGHT: %d\n", img->height);
	printf("COLOR TYPE: %d\n", img->colorType);
	printf("BIT DEPTH: %d\n", img->bitDepth);
	printf("COLOR CHANNELS: %d\n", png_get_channels(img->pngPtr,
img->infoPtr));
}
```

Название файла: cli.h

```
#pragma once

#include <unistd.h>
#include "error.h"
#include "png_structs.h"

typedef struct Configs {
	int x1Src, y1Src;
	int x2Src, y2Src;
	int xDst, yDst;
	int width;
	Pixel srcColor;
	Pixel dstColor;
	int frameType;
	double frameStyle;
	int frameInvert;
	int frameForm;
	int info;
	int cpyArea;
	int repaint;
	int circleRects;
	int drawFrame;
	char *output;
} Configs;

/* выводит справку о программе */
void printHelp();
```

```
/* возвращает заполненный значениями по умолчанию экземпляр структуры
Configs */
Configs createConfig();

/* по опции и указателю на структуру Configs обрабатывает входные данные
и заполняет структуру,
 * возвращает статус проверки (1 - требуется обработка; -1 - ошибка
чтения) */
int processOpt(int opt, Configs *config);

/* заполняет структуру пикселя из строки формата "R.G.B.<A>", возвращает
1 при некорректном вводе цвета */
int fillPixel(const char *colorString, Pixel *pix);
```

Название файла: cli.c

```
#include "cli.h"

void printHelp()
{
	puts("\nНАИМЕНОВАНИЕ");
	puts("\tcw_png - утилита для обработки PNG изображений");
	puts("СИНТАКСИС");
	puts("\tcw_png [имя исходного файла] [ключи-конфигураторы] [ключ
функции 1] [...] [опционально: имя выходного файла]");
	puts("ОПИСАНИЕ");
	puts("\tДанная утилита позволяет обработать один PNG файл
несколькими функциями поочередно. То есть одним вызовом можно применить
несколько функций, притом для каждой функции можно отдельно изменить
настройки.");
	puts("ФУНКЦИИ");
	puts("\t-d/--duplicate - копирование заданной области по
координатам левого верхнего (-f), правого нижнего (-s) углов области-
источника, а также левого верхнего (-t) угла области-получателя.");
	puts("\t-r/--repaint - замена всех пикселей изображения заданного
цвета (-c) на другой заданный цвет (-C).");
	puts("\t-l/--locate-rect - поиск всех залитых прямоугольников
заданного цвета (-c) и их обводка линиями заданного цвета (-C)
определенной ширины (-w).");
	puts("\t-b/--bezel - рисование рамки ширины (-w) (при маленьких
значениях рамка может пропасть) и цвета (-C) по краям изображения. Вид
рамки можно менять несколькими настройками:");
	puts("\t   -Тип используемого фрактала для получения узора (-T):
значение от 0 до 5. По умолчанию 0.");
	puts("\t\t-0-2 - фрактальные узоры. 1 задаёт множество Жюлиа
(хорошие узоры при стиле около 5). 2 задаёт биоморфов.");
	puts("\t\t-2-5 - узоры простых чисел. 3 - узор микросхемы, 4 -
подобие фрактальных треугольников, 5 - нечто среднее.");
	puts("\t   -Инвертирование узора (-I).");
	puts("\t   -Параметр стиля узора (-S): Допустимы вещественные
числа. Данный параметр качественно меняет узор. Каждый узор меняется по-
своему.");
	puts("\t   -Форма заливки узора (-F). Без флага рамка строится из
блоков-узоров. С флагом рамка менее однородная, поскольку получена
обрезкой одного узора.");
	puts("КОНФИГУРАЦИИ");
```

```
	puts("\t-f/--first <x.y> - координаты первой точки, требуемой
функцией. По умолчанию 0.0.");
	puts("\t-s/--second <x.y> - координаты второй точки, требуемой
функцией. По умолчанию 0.0.");
	puts("\t-t/--third <x.y> - координаты третьей точки, требуемой
функцией. По умолчанию 0.0.");
	puts("\t-w/--width <value> - задаёт толщину отрезка/рамки. По
умолчанию 1.");
	puts("\t-c/--src-col <R.G.B/R.G.B.A> - задаёт искомый цвет. По
умолчанию 0.0.0.255.");
	puts("\t-c/--dst-col <R.G.B/R.G.B.A> - задаёт конечный цвет. По
умолчанию 0.0.0.255.");
	puts("\t-S/--style <value> - рекомендуется значения от 0 до 10,
допустимы вещественные. Хорошие узоры получаются в диапазоне 7 - 9. По
умолчанию 8.");
	puts("\t-T/--type <0-5> - меняет тип узора.");
	puts("\t-I/--invert - инвертирует узор.");
	puts("\t-F/--form - рамка образуется обрезкой одного фрактального
узора, без флага рамка строится из нескольких блоков-узоров.");
	puts("ПРОЧИЕ");
	puts("\t-o/--output <имя выходного файла> - переназначает выходной
файл. По умолчанию изменяется исходный файл.");
	puts("\t-i/--info - выводит информацию об выходном PNG файле.");
	puts("\t-h/--help - выводит данную справку.");
}

Configs createConfig()
{
	Configs config = {
			.x1Src = 0, .y1Src = 0,
			.x2Src = 0, .y2Src = 0,
			.xDst = 0, .yDst = 0,
			.width = 1,
			.srcColor = {0, 0, 0, 255},
			.dstColor = {0, 0, 0, 255},
			.frameType = 0,
			.frameStyle = 8.,
			.frameInvert = 0,
			.frameForm = 0,
			.info = 0,
			.cpyArea = 0,
			.repaint = 0,
			.circleRects = 0,
			.drawFrame = 0,
			.output = NULL,
	};

	return config;
}

int processOpt(int opt, Configs *config)
{
	switch (opt)
	{
		case 'f':
			if (sscanf(optarg, "%d.%d", &config->x1Src,
&config->y1Src) != 2)
```

```
                {
                     panic("Incorrect coordinate input format");
                     return -1;
                }
                break;
           case 's':
                if (sscanf(optarg, "%d.%d", &config->x2Src,
&config->y2Src) != 2)
                {
                     panic("Incorrect coordinate input format");
                     return -1;
                }
                break;
           case 't':
                if (sscanf(optarg, "%d.%d", &config->xDst,
&config->yDst) != 2)
                {
                     panic("Incorrect coordinate input format");
                     return -1;
                }
                break;
           case 'c':
                if (fillPixel(optarg, &config->srcColor))
                {
                     panic("Incorrect color input format");
                     return -1;
                }
                break;
           case 'C':
                if (fillPixel(optarg, &config->dstColor))
                {
                     panic("Incorrect color input format");
                     return -1;
                }
                break;
           case 'w':
                if (sscanf(optarg, "%d", &config->width) != 1)
                {
                     panic("Incorrect width input format");
                     return -1;
                }
                break;
           case 'S':
                if (sscanf(optarg, "%lf", &config->frameStyle) != 1)
                {
                     panic("Incorrect frame style input format");
                     return -1;
                }
                break;
           case 'T':
                if (sscanf(optarg, "%d", &config->frameType) != 1)
                {
                     panic("Incorrect frame type input format");
                     return -1;
                }
                break;
           case 'I':
```

```
                config->frameInvert = 1;
                break;
            case 'F':
                config->frameForm = 1;
                break;
            case 'o':
                config->output = optarg;
                break;
            case 'r':
                config->repaint = 1;
                return 1;
            case 'd':
                config->cpyArea = 1;
                return 1;
            case 'l':
                config->circleRects = 1;
                return 1;
            case 'b':
                config->drawFrame = 1;
                return 1;
            case 'i':
                config->info = 1;
                break;
            case 'h':
                printHelp();
                break;
            default:
                break;
        }

    return 0;
}

int fillPixel(const char *colorString, Pixel *pix)
{
    int colorCom[4] = {-1, -1, -1, -1};
    pix->alpha = 255;

    if (sscanf(colorString, "%d.%d.%d.%d", colorCom, colorCom + 1,
colorCom + 2, colorCom + 3) < 3)
        return 1;

    for (int i = 0; i < 3; i++)
        if (colorCom[i] < 0 || colorCom[i] > 255)
            return 1;

    pix->red = colorCom[0];
    pix->green = colorCom[1];
    pix->blue = colorCom[2];
    if (colorCom[3] >= 0 && colorCom[3] <= 255) pix->alpha =
colorCom[3];

    return 0;
}
```

Название файла: png_process_support.h

```
#pragma once
```

```
#include <math.h>
#include <string.h>
#include <complex.h>
#include "png_structs.h"

/* сравнивает пиксели по каждой компоненте, возвращает 1, если равны, и 0
в обратном случае */
int pixCmp(Pixel first, Pixel second);

/* меняет значения двух переменных с известной длиной */
void swap(void *first, void *second, size_t size);

double fractalKali(double x, double y, double cx, double cy, int width,
int height);

double fractalJulia(double x, double y, double cx, double cy, int width,
int height);

double fractalBio(double x, double y, double cx, double cy, int width,
int height);

/* возвращает 1, если первое число больше второго, 0, если равны, и -1,
если второе больше первого */
int cmpGreat(double first, double second);

/* возвращает 1, если второе число больше первого, 0, если равны, и -1,
если первое больше второго */
int cmpLess(double first, double second);

/* возвращает 1, если n - простое число; 0 - если нет */
int isPrime(int n);

/* возвращает результат "исключающего или" двух входных чисел */
int XOR(int a, int b);

/* возвращает результат "побитового и" двух входных чисел */
int AND(int a, int b);

/* возвращает результат "побитового или" двух входных чисел */
int OR(int a, int b);
```

Название файла: png_process_support.c

```
#include "png_process_support.h"

int pixCmp(Pixel first, Pixel second, int CPP)
{
     if (CPP == 4)
          return first.red == second.red && first.green == second.green
&& first.blue == second.blue &&
                  first.alpha == second.alpha;
     else
          return first.red == second.red && first.green == second.green
&& first.blue == second.blue;
}

void swap(void *first, void *second, size_t size)
```

```
{
	void *buf = (void *) malloc(size);

	memcpy(buf, first, size);
	memcpy(first, second, size);
	memcpy(second, buf, size);

	free(buf);
}

double fractalKali(double x, double y, double cx, double cy, int width,
int height)
{
	double m;
	x = (2 * x - width) / width;
	y = (2 * y - height) / height;

	for (int i = 0; i < 10; i++)
	{
		x = fabs(x);
		y = fabs(y);
		m = (x * x + y * y);
		x = x / m - cx / width;
		y = y / m - cy / height;
	}

	return x + y + sqrt(x * x + y * y) / 2 > 1.5 ? 1 : 0;
}

double fractalJulia(double x, double y, double cx, double cy, int width,
int height)
{
	double complex z = (2 * y - height) / height * 1.5 + I * (2 * x -
width) / width * 1.5;
	double complex c = cy / height + I * cx / width;
	double R = (1 + sqrt(1 + 4 * cabs(c))) / 2;

	for (int i = 0; i < 32; i++)
	{
		z = z * z + c;
		if (cabs(z) > R)
			return sin(i / 5.) + sin(i / 6.) + sin(i / 7.) > 1.5 ?
1 : 0;
	}

	return 0;
}

double fractalBio(double x, double y, double cx, double cy, int width,
int height)
{
	double complex z = (2 * x - width) / width * 1.5 + I * (2 * y -
height) / height * 1.5;
	double complex c = 1.07 + I * 0.0001;

	for (int i = 0; (fabs(creal(z)) < 80 || fabs(cimag(z)) < 80 ||
cabs(z) < 80) && i < 50; i++)
```

```
            z = cpow(z, 2 + 2 * (cx / width + cy / height)) + c;

      return fabs(creal(z)) < 50 || fabs(cimag(z)) < 50 * 50 ? 1 : 0;
}

int cmpGreat(double first, double second)
{
      return first > second ? 1 : first == second ? 0 : -1;
}

int cmpLess(double first, double second)
{
      return first > second ? -1 : first == second ? 0 : 1;
}

int isPrime(int n)
{
      for (int i = 2; i <= sqrt(n); i++)
            if (n % i == 0)
                  return 0;
      return 1;
}

int XOR(int a, int b)
{
      return a ^ b;
}

int AND(int a, int b)
{
      return a & b;
}

int OR(int a, int b)
{
      return a | b;
}
```

Название файла: png_process.h

```
#pragma once

#include "png_structs.h"
#include "png_process_support.h"
#include "png_io.h"

#define CORD_BUF 5
#define CORD_IN(X1, Y1, W, H) ((X1) >= 0 && (Y1) >= 0 && (X1) < (W) &&
(Y1) < (H))

/* принимает искомый и конечный цвет, а также изображение, после чего
меняет в нём все пиксели искомого цвета на конечный */
void changeColor(Pixel src, Pixel dst, Image *img);

/* принимает координаты левого верхнего и правого нижнего угла копируемой
области, а также координаты левого верхнего угла области-получателя,
копирует заданную копируемую область в заданную */
void copyArea(int x1Src, int y1Src, int x2Src, int y2Src, int xDst, int
```

```
yDst, Image *img);

/* принимает координаты начала и конца прямого отрезка, рисует
горизонтальную или вертикальную линию */
void drawStraightLine(int x1, int y1, int x2, int y2, Pixel color, Image
*img);

/* проверяет, лежит ли точка (x, y) хотя бы в одном из прямоугольников с
координатами из cords */
int isDotInRects(int x, int y, int **cords, int cordLen);

/* принимает искомый цвет залитых прямоугольников, а также цвет и толщину
их обводки, после чего ищет все залитые прямоугольники заданного цвета и
обводит их линями заданных параметров */
void circleFilledRect(Pixel rectCol, Pixel strokeCol, int width, Image
*img);

/* расширяет изображение, прибавляя к каждой стороне по width */
void expandImg(Image *img, int width);

/* принимает указатель на формулу фрактала, компаратор, "стиль" фрактала,
форму заливки, цвет рамки, её ширину, после чего рисует соответствующую
рамку */
void drawFrameFrac(double (*frac)(double x, double y, double cx, double
cy, int width, int height),
                int (*inv)(double first, double second),
                double style, int form, Pixel color, int width, Image
*img);

/* принимает указатель на операцию с двумя числами, компаратор, "стиль",
форму заливки, цвет рамки, её ширину, после чего рисует соответствующую
рамку */
void drawFramePrime(int (*bitOp)(int a, int b), int (*inv)(double first,
double second),
                double style, int form, Pixel color, int width, Image
*img);
```

Название файла: png_process.c

```
#include "png_process.h"

void changeColor(Pixel src, Pixel dst, Image *img)
{
    for (int y = 0; y < img->height; y++)
       for (int x = 0; x < img->width; x++)
          if (pixCmp(src, getPixel(x, y, img),
png_get_channels(img->pngPtr, img->infoPtr)))
             putPixel(x, y, dst, img);
}

void copyArea(int x1Src, int y1Src, int x2Src, int y2Src, int xDst, int
yDst, Image *img)
{
    if (x1Src > x2Src) swap(&x1Src, &x2Src, sizeof(int));
    if (y1Src > y2Src) swap(&y1Src, &y2Src, sizeof(int));
    if (x1Src < 0) x1Src = 0;
    if (y1Src < 0) y1Src = 0;
    if (x1Src >= img->width) x1Src = img->width - 1;
```

```
    if (y1Src >= img->height) y1Src = img->height - 1;
    if (x2Src >= img->width) x2Src = img->width - 1;
    if (y2Src >= img->height) y2Src = img->height - 1;
    int areaWidth = x2Src - x1Src + xDst < img->width ? x2Src - x1Src +
1 : img->width - xDst;
    int areaHeight = y2Src - y1Src + yDst < img->height ? y2Src - y1Src +
1 : img->height - yDst;
    int CPP = png_get_channels(img->pngPtr, img->infoPtr); /* кол-во
компонент на пиксель */

    png_bytepp areaBuf = (png_bytepp) malloc(areaHeight *
sizeof(png_bytep));
    for (int y = y1Src; y < y1Src + areaHeight; y++)
    {
       areaBuf[y - y1Src] = (png_bytep) malloc(CPP * areaWidth *
sizeof(png_byte));
       memcpy(areaBuf[y - y1Src], img->rowPointers[y] + x1Src * CPP, CPP
* areaWidth * sizeof(png_byte));
    }

    for (int y = yDst > 0 ? yDst : 0; y < yDst + areaHeight; y++)
    {
       if (xDst > 0)
          memcpy(img->rowPointers[y] + xDst * CPP, areaBuf[y - yDst], CPP
* areaWidth * sizeof(png_byte));
       else if (-xDst < areaWidth)
          memcpy(img->rowPointers[y], areaBuf[y - yDst] - xDst * CPP, CPP
* (areaWidth + xDst) * sizeof(png_byte));
    }

    for (int i = 0; i < areaHeight; i++)
       free(areaBuf[i]);
    free(areaBuf);
}

void drawStraightLine(int x1, int y1, int x2, int y2, Pixel color, Image
*img)
{
    if (x1 > x2) swap(&x1, &x2, sizeof(int));
    if (y1 > y2) swap(&y1, &y2, sizeof(int));

    if (x1 == x2 && x1 >= 0 && x1 < img->width)
       for (int y = y1 > 0 ? y1 : 0; y <= y2 && y < img->height; y++)
          putPixel(x1, y, color, img);
    else if (y1 == y2 && y1 >= 0 && y1 < img->height)
       for (int x = x1 > 0 ? x1 : 0; x <= x2 && x < img->width; x++)
          putPixel(x, y1, color, img);
}

int isDotInRects(int x, int y, int **cords, int cordLen)
{
    for (int i = 0; i < cordLen; i++)
       if (x >= cords[i][0] && x <= cords[i][2] && y >= cords[i][1] && y
<= cords[i][3])
          return 1;

    return 0;
```

```
}

void circleFilledRect(Pixel rectCol, Pixel strokeCol, int width, Image
*img)
{
    int cordCount = 0;
    int CPP = png_get_channels(img->pngPtr, img->infoPtr);
    int xCheck, yCheck;
    int x1, y1, x2, y2;
    int **cordArr = (int **) malloc(CORD_BUF * sizeof(int *));
    for (int i = 0; i < CORD_BUF; i++)
       cordArr[i] = (int *) malloc(4 * sizeof(int));

    for (int y = 0; y < img->height; y++)
    {
       for (int x = 0; x < img->width; x++)
       {
          if (pixCmp(getPixel(x, y, img), rectCol, CPP)
&& !isDotInRects(x, y, cordArr, cordCount))
          {
             x1 = x, x2 = x;
             y1 = y, y2 = y;
             xCheck = 1, yCheck = 1;
             while (xCheck || yCheck)
             {
                if (yCheck)
                {
                   for (int i = x1; i <= x2 && yCheck; i++)
                      if (y2 + 1 >= img->height || !pixCmp(getPixel(i, y2
+ 1, img), rectCol, CPP) ||
                         isDotInRects(i, y2 + i, cordArr, cordCount))
                         yCheck = 0;
                   if (yCheck)
                      y2++;
                }
                if (xCheck)
                {
                   for (int i = y1; i <= y2 && xCheck; i++)
                      if (x2 + 1 >= img->width || !pixCmp(getPixel(x2 +
1, i, img), rectCol, CPP) ||
                         isDotInRects(x2 + 1, i, cordArr, cordCount))
                         xCheck = 0;
                   if (xCheck)
                      x2++;
                }
             }
             x = x2;

             if (cordCount && cordCount % CORD_BUF == 0)
             {
                int **tmp = (int **) realloc(cordArr, (cordCount +
CORD_BUF) * sizeof(int *));
                if (tmp == NULL)
                   crash("Memory reallocate error", img);
                cordArr = tmp;
                for (int i = cordCount; i < cordCount + CORD_BUF; i++)
                   cordArr[i] = (int *) malloc(4 * sizeof(int));
```

```
            }
            cordArr[cordCount][0] = x1, cordArr[cordCount][1] = y1,
cordArr[cordCount][2] = x2, cordArr[cordCount][3] = y2;
            cordCount++;
         }
      }
   }

   for (int i = 0; i < cordCount; i++)
   {
      x1 = cordArr[i][0], y1 = cordArr[i][1], x2 = cordArr[i][2], y2 =
cordArr[i][3];
      for (int j = width >= 0 ? 1 : 0; abs(j) <= abs(width + (width >=
0 ? 0 : 1)); j += width > 0 ? 1 : -1)
      {
         if (x1 - j <= x2 + j)
         {
            drawStraightLine(x1 - j, y1 - j, x2 + j, y1 - j, strokeCol,
img);
            drawStraightLine(x1 - j, y2 + j, x2 + j, y2 + j, strokeCol,
img);
         }
         if (y1 - j <= y2 + j)
         {
            drawStraightLine(x1 - j, y1 - j, x1 - j, y2 + j, strokeCol,
img);
            drawStraightLine(x2 + j, y1 - j, x2 + j, y2 + j, strokeCol,
img);
         }
      }
   }

   for (int i = 0; i < ((cordCount - 1) / CORD_BUF + 1) * CORD_BUF; i++)
      free(cordArr[i]);
   free(cordArr);
}

void expandImg(Image *img, int width)
{
   int CPP = png_get_channels(img->pngPtr, img->infoPtr); /* кол-во
компонент на пиксель */
   img->width += width * 2;
   img->height += width * 2;
   png_bytepp tmp = img->rowPointers;
   img->rowPointers = png_malloc(img->pngPtr, img->height *
sizeof(png_bytep));

   for (int i = 0; i < img->height; i++)
   {
      img->rowPointers[i] = png_malloc(img->pngPtr, CPP * img->width *
sizeof(png_byte));
      memset(img->rowPointers[i], 0, CPP * img->width *
sizeof(png_byte));
   }
   for (int y = width; y < img->height - width; y++)
      memcpy(img->rowPointers[y] + width * CPP, tmp[y - width], CPP *
(img->width - 2 * width) * sizeof(png_byte));
```

```
    for (int i = 0; i < img->height - 2 * width; i++)
       png_free(img->pngPtr, tmp[i]);
    png_free(img->pngPtr, tmp);
}

void drawFrameFrac(double (*frac)(double x, double y, double cx, double
cy, int width, int height),
                int (*inv)(double first, double second),
                double style, int form, Pixel color, int width, Image
*img)
{
    if (-width * 2 > img->height || -width * 2 > img->width)
       width = -img->height / 2;

    if (width > 0)
       expandImg(img, width);
    else
       width = -width;

    double fracSum;
    int fracWidth = form ? img->width : width;
    int fracHeight = form ? img->height : width;
    int balancedFracWidth;
    int balancedFracHeight = img->height - 2 * width;
    if (img->width >= fracWidth && fracWidth)
       balancedFracWidth = (img->width) / (img->width / fracWidth);
    if (img->height - 2 * width * (form ? 0 : 1) >= fracHeight &&
fracHeight)
       balancedFracHeight = (img->height - 2 * width * (form ? 0 : 1)) /
                     ((img->height - 2 * width * (form ? 0 : 1)) /
fracHeight);

    for (int x = 0; x < img->width; x++)
    {
       for (int y = 0; y < width; y++)
       {
          fracSum = frac(x % balancedFracWidth, y, balancedFracWidth *
style / 10.0, fracHeight * style / 10.0,
                      balancedFracWidth, fracHeight);
          if (inv(fracSum, 0.5) > 0)
          {
             putPixel(x, y, color, img);
             putPixel(x, img->height - y - 1, color, img);
          }
       }
    }
    for (int x = 0; x < width; x++)
    {
       for (int y = width; y < img->height - width; y++)
       {
          if (form)
             fracSum = frac(x, y, fracWidth * style / 10.0, fracHeight *
style / 10.0, fracWidth, fracHeight);
          else
             fracSum = frac((y - width) % balancedFracHeight, x,
balancedFracHeight * style / 10.0,
```

```
                           fracWidth * style / 10.0, balancedFracHeight,
fracWidth);
          if (inv(fracSum, 0.5) > 0)
          {
             putPixel(x, y, color, img);
             putPixel(img->width - x - 1, y, color, img);
          }
       }
    }
}

void drawFramePrime(int (*bitOp)(int a, int b), int (*inv)(double first,
double second),
                double style, int form, Pixel color, int width, Image
*img)
{
    if (-width * 2 > img->height || -width * 2 > img->width)
       width = -img->height / 2;

    if (width > 0)
       expandImg(img, width);
    else
       width = -width;

    int patternWidth = form ? img->width : width;
    int patternHeight = form ? img->height : width;
    int balancedPatternWidth = 0;
    int balancedPatternHeight = img->height - 2 * width;
    if (img->width >= patternWidth && patternWidth)
       balancedPatternWidth = (img->width) / (img->width / patternWidth);
    if (img->height - 2 * width * (form ? 0 : 1) >= patternHeight &&
patternHeight)
       balancedPatternHeight = (img->height - 2 * width * (form ? 0 : 1))
/
                           ((img->height - 2 * width * (form ? 0 : 1)) /
patternHeight);

    for (int x = 0; x < img->width; x++)
    {
       for (int y = 0; y < width; y++)
       {
          if (inv(isPrime(bitOp((int) ((x % balancedPatternWidth) * style
/ 10), (int) (y * style / 10))), 0.5) >
              0)
          {
             putPixel(x, y, color, img);
             if (form == 0)
                putPixel(x, img->height - width + y, color, img);
          }
          if (form && inv(isPrime(bitOp((int) ((x % balancedPatternWidth)
* style / 10),
                                    (int) ((img->height - y - 1) * style /
10))), 0.5) > 0)
             putPixel(x, img->height - y - 1, color, img);
       }
    }
    for (int x = 0; x < width; x++)
```

```
    {
       for (int y = width; y < img->height - width; y++)
       {
          if (inv(isPrime(bitOp((int) ((y % balancedPatternHeight) *
style / 10),
                          (int) (x * style / 10))), 0.5) > 0)
          {
             putPixel(x, y, color, img);
             if (form == 0)
                putPixel(img->width - width + x, y, color, img);
          }
          if (form && inv(isPrime(bitOp((int) ((y %
balancedPatternHeight) * style / 10),
                                (int) ((img->width - x - 1) * style /
10))), 0.5) > 0)
             putPixel(img->width - x - 1, y, color, img);
       }
    }
}
```

Название файла: Makefile

```
CC=gcc
CFLAGS=-c -Wall -lpng -lm

all: cw_png clean

cw_png: main.o png_io.o png_structs.o png_process_support.o
png_process.o cli.o error.o
     $(CC) main.o png_io.o png_structs.o png_process_support.o
png_process.o cli.o error.o -o cw_png -lpng -lm

main.o: main.c png_io.h png_structs.h cli.h error.h
     $(CC) $(CFLAGS) main.c

error.o: error.c png_structs.h
     $(CC) $(CFLAGS) error.c

png_io.o: png_io.c error.h png_structs.h
     $(CC) $(CFLAGS) png_io.c

png_structs.o: png_structs.c
     $(CC) $(CFLAGS) png_structs.c

png_process_support.o: png_process_support.c png_structs.h
     $(CC) $(CFLAGS) png_process_support.c

cli.o: cli.c error.h png_structs.h
     $(CC) $(CFLAGS) cli.c

png_process.o: png_process.c png_process_support.h png_structs.h png_io.h
     $(CC) $(CFLAGS) png_process.c

clean:
     rm *.o
```