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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка изображений

Студент гр. 2383 _____ Исаев Э.Н.

Преподаватель _____ Азаревич А.Д.

Санкт-Петербург

2023

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Исаев Э.Н.

Группа 2383

Тема работы: Работа с изображениями в языке Си

Исходные данные:
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. Рисование квадрата. Квадрат определяется:

- Координатами левого верхнего угла
- Размером стороны
- Толщиной линий
- Цветом линий
- Может быть залит или нет
- Цветом которым он залит, если пользователем выбран залитый

2. Поменять местами 4 куска области. Выбранная пользователем прямоугольная область делится на 4 части и эти части меняются местами. Функционал определяется:

- Координатами левого верхнего угла области
- Координатами правого нижнего угла области
- Способом обмена частей: “по кругу”, по диагонали

3. Находит самый часто встречаемый цвет и заменяет его на другой заданный цвет. Функционал определяется Цветом, в который надо перекрасить самый часто встречаемый цвет.
4. Инверсия цвета в заданной области. Функционал определяется
   - Координатами левого верхнего угла области
   - Координатами правого нижнего угла области

Содержание пояснительной записки:
«Содержание», «Введение», «Ход выполнения работы», «Заключение», «Список использованных источников», «Приложение А», «Приложение Б».

Предполагаемый объем пояснительной записки:
Не менее 15 страниц.

Дата выдачи задания: 22.03.2023

Дата сдачи реферата: 25.05.2023

Дата защиты реферата: 27.05.2023

Студент _____ Исаев Э.Н.

Преподаватель _____ Азаревич А.Д.

АННОТАЦИЯ

Курсовая работы предполагает написание программы на языке СИ по обработке изображений формата PNG. Взаимодействие с программой происходит за счет CLI (Command Line Interface). В зависимости от запроса пользователя программа должна обработать соответствующее изображение и сохранить его. Для решения поставленных задач были применены методы работы с динамической памятью в СИ, функции стандартной библиотеки, а также функции библиотеки *libpng* и *getopt.h*. В результате была написана программа, выполняющая все поставленные задачи и соответствующая указанным требованиям.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы.

Целью данной работы является создание программы, которая обрабатывает изображения PNG формата в соответствии с запросом пользователя

Задачи.

Для достижения поставленной цели требуется решить следующие задачи:

- Реализовать считывание и запись PNG-изображения
- Реализовать рисование при помощи алгоритмов взаимодействия с памятью
- Обработать исключения
- Реализовать командный интерфейс программы

1. ОСНОВНАЯ ОБРАБОТКА

1.1. Включения и структуры

Для реализации всех функций в программе, а также ее корректной работы включаются заголовочные файлы стандартной библиотеки языка СИ: *<stdlib.h>*, *<stdio.h>*, *<stdbool.h>*, *<png.h>*, *<getopt.h>*. Так же используется директива *#pragma once*, необходимая, чтобы заголовочные файлы включались только один раз.

В данной курсовой работе определены две структуры: *Png* и *png_color_struct_amount.*

Стурктура *Png* содержит поля: *width* — ширина изображения в пикселях, *height* — высота изображения в пикселях, *color_type* — поле типа *png_byte*, определяющее тип цветовой модели изображения (далее в курсовой работе используется тип RGBA), *bit_depth* — количество битов на каждый канал пикселя (глубина цвета), *png_ptr* — указатель на структуру *png_struct*, используемую библиотекой *libpng* для чтения или записи PNG-изображений, *info_ptr* — указатель на структуру *png_info*, которая содержит информацию о PNG-изображении, **row_pointers* — указатель на массив указателей *png_bytep*, указывающих на массив байтов, представляющих значения пикселей в соответствующей строке изображения.

Структура *png_color_struct_amount* представляет из себя модификацию структуры *png_color_struct*, и содержит поля цвета (принимают значение от 0 до 255, что связано с глубиной цвета): *red* — поле типа *png_byte*, определяющее значение красного компонента цвета, *green* — поле типа *png_byte*, определяющее значение красного компонента цвета, *blue* — поле типа *png_byte*, определяющее значение красного компонента цвета, - а также поле *amount* — целочисленное поле, определяющее количество пикселей заданного цвета.

1.2. Чтение и запись изображения

Для обработки изображения, вначале необходимо его считать, а после записать в PNG-файл. Считывание изображения осуществляется при помощи функций *read_png_file()*, запись — *write_png_file()*.

Функция *read_png_file()* предназначена для чтения PNG-файла и заполнения структуры Png с соответствующей информацией о изображении. На вход она принимает: *file_name*: указатель на строку, содержащую имя PNG-файла, который требуется прочитать, *image*: указатель на структуру Png, в которую будет сохранена информация о прочитанном изображении.

После чего открывается PNG-файл с помощью функции fopen в режиме «rb» — чтение двоичного файла. Если не удалось открыть файл, выводится сообщение об ошибке, и функция возвращает 0.

Далее создается структура png_struct с помощью функции *png_create_read_struct(),* которая будет использоваться для чтения PNG-файла. Если не удалось создать структуру, файл закрывается, выводится сообщение об ошибке, и функция возвращает 0. А также создается структура png_info с помощью функции *png_create_info_struct()*, содержащая информацию об изображении. Если не удалось создать структуру, файл закрывается, удаляется структура чтения PNG с помощью функции *png_destroy_read_struct()* и выводится сообщение об ошибке. Затем функция возвращает 0.

После устанавливается точка возврата *setjmp* для обработки возможных ошибок во время инициализации чтения PNG-файла. Если происходит ошибка, файл закрывается, удаляется структура чтения PNG и информации PNG, и выводится сообщение об ошибке. Затем функция возвращает 0. Происходит сама инициализация ввода/выводы для структуры чтения PNG с помощью функции *png_init_io()*.

Читается информация о PNG-изображении с помощью функции *png_read_info()*. Информация о ширине, высоте, типе цвета и глубине битов сохраняется в соответствующих полях структуры Png.

Обрабатываемые условия:

- Если глубина битов цвета равна 16, вызывается функция *png_set_strip_16()* для преобразования глубины битов в 8.
- Если тип цвета — палитра, вызывается функция *png_set_palette_to_rgb()* для преобразования палитры в RGB.
- Если PNG-изображение содержит прозрачность (tRNS), вызывается функция *png_set_tRNS_to_alpha()* для преобразования альфа-канала.
- Если тип цвета — RGB, оттенки серого или палитра, вызывается функция *png_set_filler()* для установки заполнителя на 0xFF (полностью непрозрачный) после каждого пикселя.
- Если тип цвета — оттенки серого или оттенки серого с альфа-каналом, вызывается функция *png_set_gray_to_rgb()* для преобразования в RGB.

После условий обновляется информация об изображении с помощью функции *png_read_update_info()*. А также динамически выделяется память для массива указателей строк *row_pointers* с размером, равным высоте изображения.

С помощью цикла для каждой строки изображения выделяется память, достаточная для хранения байтов пикселей, с помощью функции *malloc()* и *png_get_rowbytes()*.

В конце считывается изображение с помощью функции *png_read_image()* и файл закрывается. В результате вся необходимая информация об изображении сохраняется в структуру Png.

Функция *write_png_file()* предназначена для записи данных об изображении из структуры Png в файла. На вход она принимает: *file_name*: указатель на строку, содержащую имя PNG-файла, в который будет записана струтура, *image*: указатель на структуру Png, которая непосредственно содержит данные об изображении.

После чего открывается PNG-файл с помощью функции fopen в режиме «rb» — записи двоичного файла. Если не удалось открыть файл, выводится сообщение об ошибке, и функция возвращает 0.

Далее создается структура png_struct с помощью функции *png_create_write_struct(),* которая будет использоваться для записи PNG-файла. Если не удалось создать структуру, файл закрывается, выводится сообщение об ошибке, и функция возвращает 0. А также создается структура png_info с помощью функции *png_create_info_struct()*, которая будет содержать информацию об изображении. Если не удалось создать структуру, файл закрывается, удаляется структура чтения PNG с помощью функции *png_destroy_write_struct()* и выводится сообщение об ошибке. Затем функция возвращает 0.

После устанавливается точка возврата *setjmp* для обработки возможных ошибок во время инициализации записи PNG-файла. Если происходит ошибка, файл закрывается, удаляется структура записи PNG и информации PNG, и выводится сообщение об ошибке. Затем функция возвращает 0. Происходит сама инициализация ввода/выводы для структуры записи PNG с помощью функции *png_init_io()*.

Далее устанавливаются параметры изображения в header с помощью функции png_set_IHDR. Здесь задаются ширина, высота, глубина битов, тип цвета и другие характеристики изображения. После чего читается информация об изображении с помощью функции *png_write_info()*.

В конце происходит сама запись изображения с помощью функции *png_write_image()* и завершается запись с помощью функции *png_write_end()*. В результате вся необходимая информация об изображении сохраняется в структуру Png. Далее происходит освобождение выделенной памяти при помощи функции *png_destroy_write_struct(),* а также прохода циклом по строкам изображения и освобождения памяти выделенной под них и под указатель на них используя функцию *free()*. Файл закрывается. В результате вся необходимая информация об изображении из структуры записывается в файла, что позволяет сохранить изображение.

1.3. Вспомогательные функции

В ходе выполнения курсовой работы стало необходимым написать вспомогательные функции, упрощающие работу с кодом и его чтение.

Функция *swap_arrays()* предназначена для обмена значениями между двумя массивами типа png_byte. Функция принимает два указателя на массивы *arr1* и *arr2*, представляющих из себя пиксели и за счет цикла выполняет обмен значениями элементов этих массивов (компонентов пикселя).

Функция *valid_coor()* необходима для проверки валидности координат относительно размеров изображения. Если все значения допустимы для изображения с данными параметрами, то функция возвращает True, в противном случае — False.

Функция *set_pixel_color()* используется для установки пикселя структуры изображения, находящегося на заданных координатах в соответствующий цвет с полной непрозрачностью. Сначала координаты пикселя проверяются с помощью функции *valid_coor()*, после чего используя указатель на начало пиксельного массива, каждый элемент этого массива устанавливается в соответствующее пол переданного цвета *color*.

Функция *set_pixel()* используется для установки пикселя изображения, находящегося на заданных координатах в соответствующий пиксель из другой структуры изображения. Сначала координаты каждого из двух пикселей проверяются с помощью функции *valid_coor()*, если функция возвращает False, то никаких изменений не происходит — функция завершает работу. После с помощью цикла каждой компоненте пикселя первой структуры в нужной координате присваивается соответствующее значение пикселя другой структуры, так же подобранного по координате.

Функция *canvas()* используется для создания и инициализации структуры Png в виде «холста» определенного размера и цвета. Функция принимает указатель на структуру Png *image*, тип цвета *color_type*, глубину цвета *bit_depth*, высоту *height* и ширину *width* холста, а также структуру png_color для установки заданного цвета холста. Изначально происходит инициализация холста с

соответствующими параметрами, также выделяется память для хранения строк пикселей изображения. После чего при помощи цикла и функции *set_pixel_color()* все пиксели холста устанавливаются в заданный цвет *color*.

Функция *crop()* необходима, чтобы обрезать переданное ей изображение в соответствии с переданными координатами области, которую нужно сохранить. Функция создает новую структуру Png *copy*, копирует в нее исходное изображение с помощью *memcpy()*, а затем обновляет исходную структуру *img*, делая ее холстом размером под заданные координаты. После с помощью цикла и функции *set_pixel()* пиксели из копии исходного изображения присваиваются пикселям холста — *img*, структуры, которая будет использована в качестве изображения-результата.

Функция *paste()* предназначена для вставки содержимого исходного изображения *src* в изображение *des*, заданного координатами. Функция перебирает пиксели исходного изображения *src* и устанавливает их компоненты в соответствующие изображения *des*. Координаты пикселя в исходном изображении остаются неизменными, а координаты пикселя в целевом изображении рассчитываются как (j - x, i — y), чтобы учесть смещение, где i — строка, j — столбец.

1.4. Основные функции

В курсовой работе реализованы четыре основных функций работы с изображением, соответствующие требуемым.

Функция *draw_square()* используется для рисования квадрата на изображении *image*. Квадрат определяется начальными координатами верхнего левого угла (x0, y0), размером *size*, толщиной линии *line_width*, цветом линии *line_color,* флагом *is_fill* для определения, должен ли квадрат быть заполнен, и цветом заполнения *fill_color*.

Обрабатываемые условия:

- Проверяется, соответствует ли размер квадрата *size* исходному изображению *image*. Если квадрат не помещается в изображение, выводится сообщение об ошибке и функция завершается.
- Проверяется, не является ли толщина линии *line_width* слишком большой по сравнению с размерами изображения *image*. Если толщина линии слишком велика, выводится сообщение об ошибке и функция завершается.

После обработанных условия вычисляются координаты правого нижнего угла квадрата (x1, y1) на основе начальных координат (x0, y0) и размера *size*.

Рисуется левая и правая сторона квадрата путем итерации по строкам — i от y0 до y1 и установки ряда пикселей линии толщиной *line_width* в цвет *line_color* с помощью функции *set_pixel_color().*

Рисуется верхняя и нижняя сторона квадрата путем итерации по столбцам — j от x0 до x1 и установки ряда пикселей линии толщиной *line_width* в цвет *line_color* с помощью функции *set_pixel_color().*

Если установлен флаг *is_fill*, то заполняется внутренняя часть квадрата, исключая границы, путем итерации по строкам — i от y0 + *line_width* до y1 - *line_width* и столбцам — j от x0 + *line_width* до x1 - *line_width*, и установки пикселей в цвет *fill_color* с помощью функции *set_pixel_color()*.

Следующая функция *swap_pix()* используется, чтобы поменять местами четыре области изображения, что примечательно, обмен может происходить двумя способами: по кругу и диагоналями.

Обрабатываемые условия:

- Проверяются условия для корректности координат области обмена пикселями. Если условия не выполняются, выводится сообщение об ошибке, указывающее, что координаты не соответствуют размеру изображения. В таком случае функция прекращает работу .
- Проверяется нечетность размеров сторон области обмена пикселями. Если одна из сторон имеет нечетную длину, выводится предупреждающее сообщение, указывающее, что некоторые части изображения не будут

использоваться для обмена пикселями. Однако выполнение функции продолжается.

- Проверяется режим обмена пикселями (передается в качестве аргумента *mode*). Если режим не равен 1 или 2, выводится сообщение об ошибке, указывающее на отсутствие такого режима, и функция прекращает работу.

После обработанных условия вычисляются координаты медиан области обмена пикселями. Каждая медиана вычисляется как половина длины стороны области обмена пикселями.

После с помощью цикла перебираются все пиксели в заданной области. Пределы цикла по строкам равны y0 и *v_med*, пределы цикла по столбцам соответственно равны x0 и *h_med*. В каждой итерации внутреннего цикла выполняется разбиение области обмена пикселями на четыре части, и для каждой части получаются указатели на соответствующие пиксели. В зависимости от режима обмена пикселями — mode, происходит обмен пикселями между указателями. Если *mode* равен 1, происходит круговой обмен пикселями, если *mode* равен 2, происходит диагональный обмен пикселями.

Функция *change_color()* реализует замену часто встречаемого цвета в изображении на заданный. Важно условиться, что если разные часто встречаемые цвета количественно встречаются одинаково, то замена произойдет только с последним. Изначально создается массив цветов, хранящий структуры png_colora, описанные ранее, точный размер неизвестен. С помощью циклов перебираются все пиксели изображения, при каждой итерации происходит проверка, содержится ли текущий цвет пикселя в массиве цветов, если уже есть, то счетчик *amount* данного цвета увеличивается на 1, если цвет в массиве найден не был, то он добавляется в массив, после чего счетчик размерности массива *c* увеличивается на 1.

После завершения первого цикла создается цикл, который находит цвет с наибольшим количеством повторений в массиве *arr*. Для этого перебираются все элементы массива *arr*, и если *amount* текущего цвета больше значения *max*,

значения *max* обновляется, а компоненты цвета (*red*, *green*, *blue*) сохраняются в переменной *color_need_to_change*.

Последний двойной цикл перебирает все пиксели изображения еще раз. В каждой итерации проверяется, соответствует ли текущий пиксель цвету *color_need_to_change*. Если да, то компоненты цвета пикселя заменяются на компоненты цвета, переданные в аргументе *color*.

И последняя функция *inversion(),* предназначенная для инвертирования цвета изображения в заданной области.

Обрабатываемые условия:

- Проверяются условия для корректности координат области (x0, y0, x1, y1). Если условия не выполняются, выводится сообщение об ошибке, указывающее, что координаты не соответствуют размеру изображения. В таком случае функция прекращает свою работу и возвращает управление.

Задается двойной цикл, перебирающий пиксели изображения в заданной области, итерация по столбцам происходит с учетом кратности 4, так как каждый пиксель представляет четыре компоненты, три из которых в данной функции меняются, согласно формуле: <255 — ptr_value>, которая объясняется тем, что получение инвертированного цвета обусловлено вычитанием из максимального значения компоненты текущего ее значения.

3. ВЗАИМОДЕЙСТВИЕ С ПРОГРАММОЙ

3.1. Соглашения

Взаимодействие с программой происходит за счет *CLI (Command Line Interface).* Все необходимые флаги и аргументы считываются вначале, после подаются названия файлов для считывания и записи PNG-изображения соответственно.

Для использования функционала программы необходимо передать аргументы через командную строку. Доступные команды (опции) указываются после названия программы и могут быть следующими: *--draw, --swap, --mode, --change, --inversion, --size, --start, --line, --outline, --fill, --color, --help.*

В случае неправильно поданных параметров пользователь будет проинформирован о невалидности того или иного аргумента, и программа завершит работу. Если существует ошибка в написании флага, то это приведет к завершению работы программы и выводу справочной информации — *help* запрос. В использовании некоторых функций для обработки изображения пользователь должен понимать, если он использует специфичные входные параметры, что обработка будет произведена согласно его запросу с некоторыми потерями или же нестыковками.

Перед использованием функции *read_png_file(),* убедитесь, что передали правильные аргументы командной строки, и функция будет считывать изображение из файла с указанным именем и заполнять структуру struct Png данными изображения. Входной файл должен быть строго в формате PNG. Функции *read_png_file()* и *write_png_file()* работают только с 24-битными изображениями в RGB. В случае другого цветового пространства, функция вернет 0. В случае возникновения ошибок при чтении или записи PNG-изображения, функции *read_png_file()* и *write_png_file()* будут возвращать 0.

3.2. CLI

Интерфейс программы, написанной в этой работе, реализован командно — *CLI* при помощи библиотеки *getopt.h.* Работа с CLI представлена в файле main.c.

Для начала были объявлены и инициализированы переменные, соответствующие флагам функций: *f_draw, f_swap, f_change, f_inversion*. Далее в функции main() представлена основная часть обработки командой строки и вызова функций для работы с изображением.

Для начала объявляются различные переменные, которые будут использоваться в программе, включая структуру Png для хранения изображения в PNG-формате, а также переменные для хранения цветов и других параметров операций.

Также определяется массив *long_options*, который используется для определения длинных опций командной строки, таких как --draw или —swap, каждая из которых имеет свой флаг и определенный символ.

После при помощи цикла происходит обработка аргументов командной строки с помощью функции *getopt_long()* происходит обработка каждой опции. Внутри каждого блока кода определяется, какие значения переданы в опцию, и соответствующие переменные и флаги устанавливаются в нужные значения. Каждый аргумент анализируется, и соответствующая операция выполняется в зависимости от опции.

Далее в условном блоке происходит выполнение операций над изображением в зависимости от установленных флагов. Например, если флаг *f_draw* установлен в значение 1, то вызывается функция *draw_square()*, которая рисует квадрат на изображении.

После, если остались необработанные аргументы командной строки, то они поочередно выводятся на экран. И программа завершается со статусом «успешно».

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной курсовой работы была успешно достигнута ее основная цель - создание программы, способной обрабатывать изображения PNG-формата в соответствии с запросами пользователей. Для достижения этой цели были решены несколько задач.

В первую очередь была реализована функциональность считывания и записи PNG-изображений. Это включало в себя разработку соответствующих алгоритмов, способных корректно считывать данные изображений и записывать в них внесенные изменения. Благодаря этому, пользователь имеет возможность загружать изображения в программу и сохранять обработанные результаты.

Далее были реализованы алгоритмы для рисования на изображении. Эти алгоритмы взаимодействуют с памятью, позволяя пользователю редактировать изображение, добавлять новые элементы или изменять существующие.

В процессе разработки программы было уделено внимание обработке исключительных ситуаций. Это обеспечивает стабильность и надежность программы, а также предоставляет пользователю сообщения об ошибках, если таковые возникают.

Наконец, для обеспечения удобства использования был реализован командный интерфейс программы, предоставляющий удобный способ взаимодействия с программой.

Таким образом, в результате выполнения данной курсовой работы была разработана программа, способная обрабатывать изображения PNG-формата в соответствии с запросами пользователей. Программа обладает функциональностью считывания и записи изображений, алгоритмами рисования и командным интерфейсом.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Брайан Керниган, Деннис Ритчи. Язык программирования С. — Москва: Вильямс, 2015. — 304 с.
2. Веб-сайт Основы программирования на языках Си и С++ для начинающих. URL: http://cppstudio.com/
3. Веб-сайт Программирование на С и С++. URL: https://c-cpp.ru/
4. Вопрос- ответ форум. URL: https://stackoverflow.com/

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

```
#include "read_and_write.h"
#include "functions.h"

static int f_draw;
static int f_swap;
static int f_change;
static int f_inversion;
static int f_function;

int main(int argc, char *argv[]) {
    struct Png image;
    int r = 0, g = 0, b = 0;

    png_color fill_color = {0};
    png_color outline = {0};
    int start[2] = {0};
    int size = 0;
    int line = 0;
    int is_fill = 0;

    int swap[4] = {0};
    int mode = 0;

    png_color color = {0};

    int inversion_coor[4] = {0};
    png_color color1 = {0};

    static struct option long_options[] = {
            {"draw",      no_argument, &f_draw,    1},
            {"swap",      required_argument, NULL, 'w'},
            {"mode",      required_argument, NULL, 'm'},
            {"change",    required_argument, NULL, 'c'},
            {"inversion", required_argument, NULL, 'i'},
            {"size",      required_argument, NULL, 'd'}, //side's size
            {"start",     required_argument, NULL, 's'},
            {"line",      required_argument, NULL, 'l'},
            {"outline",   required_argument, NULL, 'o'}, //line_color
            {"fill",      required_argument, NULL, 'f'},
            {"help",      no_argument,       NULL, 'h'},
            {"func",      no_argument,       NULL, 'z'},
            {"color",     required_argument, NULL, 'v'},
            {NULL, 0,                        NULL, 0}
    };

    int opt;
    int option_index = 0;

    while ((opt = getopt_long(argc, argv, "w:m:c:i:s:d:l:o:f:h?",
long_options, &option_index)) != -1) {
        switch (opt) {
            case 0:
```

```
                if (long_options[option_index].flag != 0) {
                    break;
                }

            case 'w':
                sscanf(optarg, "%d,%d:%d,%d", &swap[0], &swap[1],
&swap[2], &swap[3]);
                f_swap += 1;
                break;
            case 'm':
                mode = atoi(optarg);
                break;

            case 'i':
                f_inversion += 1;
                sscanf(optarg, "%d,%d:%d,%d", &inversion_coor[0],
&inversion_coor[1], &inversion_coor[2],
                       &inversion_coor[3]);
                break;

            case 'c':
                f_change += 1;
                sscanf(optarg, "%d,%d,%d", &r, &g, &b);
                color.red = r;
                color.green = g;
                color.blue = b;
                break;

            case 's':
                sscanf(optarg, "%d,%d", &start[0], &start[1]);
                break;

            case 'd':
                size = atoi(optarg);
                break;

            case 'l':
                line = atoi(optarg);
                break;

            case 'o':
                sscanf(optarg, "%d,%d,%d", &r, &g, &b);
                outline.red = r;
                outline.green = g;
                outline.blue = b;
                break;

            case 'f':
                sscanf(optarg, "%d,%d,%d", &r, &g, &b);
                fill_color.red = r;
                fill_color.green = g;
                fill_color.blue = b;
                is_fill = 1;
                break;

            case 'z': {
                f_function += 1;
```

```
                break;
            }
            case 'v':
                sscanf(optarg, "%d,%d,%d", &r, &g, &b);
                color1.red = r;
                color1.green = g;
                color1.blue = b;
                break;

            case 'h':
            case '?':
                printHelp();
                return 0;

            default:
                printf("Unknown option %c\n", opt);
                exit(EXIT_FAILURE);
        }
    }

    read_png_file(argv[optind++], &image);
    if (f_draw) {
        draw_square(&image, start[0], start[1], size, line, outline,
is_fill, fill_color);
    }
    if (f_swap) {
        swap_pix(&image, mode, swap[0], swap[1], swap[2], swap[3]);
    }
    if (f_change) {
        change_color(&image, color);
    }
    if (f_inversion) {
        inversion(&image, inversion_coor[0], inversion_coor[1],
inversion_coor[2], inversion_coor[3]);
    }
    if (f_function) {
//        ADD FUNC HERE
//        for example
//        crop(&copy,0,0,40,40);
//        read_png_file("kub.png",&image1);

//        write_png_file("resultat.png",&result);
    }
    write_png_file(argv[optind++], &image);

    if (optind < argc) {
        printf("non-option: ");
        while (optind < argc) {
            printf("%s ", argv[optind++]);
        }
        printf("\n");
    }

    exit(EXIT_SUCCESS);
}
```

Название файла: read_and_write.c

```
#include "read_and_write.h"

int read_png_file(char *file_name, struct Png *image) {
    FILE *fp = fopen(file_name, "rb");
    if (!fp) {
        fprintf(stderr, "Error opening file %s\n", file_name);
        return 0;
    }

    image->png_ptr = png_create_read_struct(PNG_LIBPNG_VER_STRING, NULL,
NULL, NULL);
    if (!image->png_ptr) {
        fclose(fp);
        fprintf(stderr, "Error creating read struct\n");
        return 0;
    }

    image->info_ptr = png_create_info_struct(image->png_ptr);
    if (!image->info_ptr) {
        fclose(fp);
        png_destroy_read_struct(&image->png_ptr, NULL, NULL);
        fprintf(stderr, "Error creating info struct\n");
        return 0;
    }

    if (setjmp(png_jmpbuf(image->png_ptr))) {
        fclose(fp);
        png_destroy_read_struct(&image->png_ptr, &image->info_ptr, NULL);
        fprintf(stderr, "Error during init_io\n");
        return 0;
    }

    png_init_io(image->png_ptr, fp);
    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 (image->bit_depth == 16) {
        png_set_strip_16(image->png_ptr);
    }

    if (image->color_type == PNG_COLOR_TYPE_PALETTE) {
        png_set_palette_to_rgb(image->png_ptr);
    }


    if (png_get_valid(image->png_ptr, image->info_ptr, PNG_INFO_tRNS)) {
        png_set_tRNS_to_alpha(image->png_ptr);
    }
```

```
    if (image->color_type == PNG_COLOR_TYPE_RGB || image->color_type ==
PNG_COLOR_TYPE_GRAY ||
        image->color_type == PNG_COLOR_TYPE_PALETTE) {
        png_set_filler(image->png_ptr, 0xFF, PNG_FILLER_AFTER);
    }

    if (image->color_type == PNG_COLOR_TYPE_GRAY || image->color_type ==
PNG_COLOR_TYPE_GRAY_ALPHA) {
        png_set_gray_to_rgb(image->png_ptr);
    }

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

    image->row_pointers = (png_bytep *) malloc(sizeof(png_bytep) *
image->height);
    for (int 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);
}


int write_png_file(char *file_name, struct Png *image) {
    FILE *fp = fopen(file_name, "wb");
    if (!fp) {
        printf("File %s could not be opened for writing\n", file_name);
        return 0;
    }

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

    if (!image->png_ptr) {
        printf("png_create_write_struct failed\n");
        fclose(fp);
        return 0;
    }

    image->info_ptr = png_create_info_struct(image->png_ptr);
    if (!image->info_ptr) {
        printf("png_create_info_struct failed\n");
        png_destroy_write_struct(&image->png_ptr, NULL);
        fclose(fp);
        return 0;
    }

    if (setjmp(png_jmpbuf(image->png_ptr))) {
        printf("Error during init_io\n");
        png_destroy_write_struct(&image->png_ptr, &image->info_ptr);
        fclose(fp);
        return 0;
    }
```

```
    png_init_io(image->png_ptr, fp);

    png_set_IHDR(image->png_ptr, image->info_ptr, image->width,
image->height,
                 image->bit_depth, image->color_type, PNG_INTERLACE_NONE,
                 PNG_COMPRESSION_TYPE_BASE, PNG_FILTER_TYPE_BASE);

    png_write_info(image->png_ptr, image->info_ptr);
    png_write_image(image->png_ptr, image->row_pointers);
    png_write_end(image->png_ptr, NULL);

    /* cleanup heap allocation */
    png_destroy_write_struct(&image->png_ptr, &image->info_ptr);
    for (int y = 0; y < image->height; y++)
        free(image->row_pointers[y]);
    free(image->row_pointers);

    fclose(fp);
}
```

Название файла: read_and_write.h

```
#include "libs_and_strucs.h"
#pragma once
int read_png_file(char *file_name, struct Png *image);
int write_png_file(char *file_name, struct Png *image);
```

Название файла: functions.c

```
#include "functions.h"


void printHelp() {
    printf("\nPossibly options:\n");
    printf("\nDrawing a square.\n");
    printf("--draw - Command to draw. A square is defined:\n");
    printf("--start <start coordinates> - The coordinates of the upper
left corner;\n");
    printf("--size <value> - The size of the side;\n");
    printf("--line <value> - The thickness of the lines;\n");
    printf("--outline <color values> - The color of the lines;\n");
    printf("--fill <color values> - Filled color (optional).\n");
    printf("\nSwap four pieces of the target area.\n");
    printf("--swap <start coordinates>:<end coordinates> - Command to
swap. Swap is defined:\n");
    printf("--mode <value> - The method of swaping pieces: \"in a
circle\" - <1>, \"diagonally\" - <2>.\n");
    printf("\nThe most common color is replaced by another.\n");
    printf("--change <color values> - Command to change color.\n");
    printf("\nColor inversion in the target area.\n");
    printf("--inversion <start coordinates>:<end coordinates> - Command
to inverse.\n");
    printf("\nThe last two options are the names of PNG files for reading
and writing.\n");
    printf("Please be careful with the spaces and observe the option
formats.\n\n");
```

```
    printf("Examples of use:\n"
           "--draw --start 0,0 --size 100 --line 4 --outline 255,0,0 --
fill 0,255,0 test.png test_res.png\n"
           "--swap 10,20:100,200 --mode 2 test.png test_res.png\n"
           "--change 0,128,128 test.png test_res.png\n"
           "--inversion 0,0:700,700 test.png test_res.png\n\n");
}


void draw_square(struct Png *image, int x0, int y0, int size, int
line_width, png_color line_color, int is_fill,
                 png_color fill_color) {
    if (((size - x0) > image->width) || ((size - y0) > image->height)) {
        printf("the size of square do not match the size of the
image\n");
        return;
    }

    if ((line_width * 2 > image->height) || (line_width * 2 >
image->width)) {
        printf("line width is too big");
        return;
    }
// Calculate coordinates of bottom right corner of square
    int x1 = x0 + size - 1;
    int y1 = y0 + size - 1;
// Draw left and right line of square
    for (int i = y0; i <= y1; i++) {
        for (int j = 0; j < line_width; j++) {
            set_pixel_color(image, x0 + j, i, line_color);
            set_pixel_color(image, x1 - j, i, line_color);
        }
    }
// Draw top and bottom line of square
    for (int i = 0; i < line_width; i++) {
        for (int j = x0; j <= x1; j++) {
            set_pixel_color(image, j, y0 + i, line_color);
            set_pixel_color(image, j, y1 - i, line_color);
        }
    }
// Fill square if requested
    if (is_fill) {
        for (int i = y0 + line_width; i <= y1 - line_width; i++) {
            for (int j = x0 + line_width; j <= x1 - line_width; j++) {
                set_pixel_color(image, j, i, fill_color);

            }
        }
    }
}


void swap_pix(struct Png *image, int mode, int x0, int y0, int x1, int
y1) {
    if (((x1 - x0) >= image->width) || ((y1 - y0) >= image->height) ||
(x1 >= image->width) || (y1 >= image->height)) {
        printf("the coordinates do not match the size of the image\n");
```

```
        return;
    }
    // Oddness of the sides of the area
    if ((x1 - x0 + 1) % 2 != 0 || (y1 - y0 + 1) % 2 != 0) {
        printf("some parts of the image were not used for the change\n");
    }

    if ((mode != 1 && mode != 2)) {
        printf("no such mode");
        return;
    }
    // Median
    int h_med = (x1 - x0) / 2;
    int v_med = (y1 - y0) / 2;

    for (int i = y0; i <= v_med; i++) {
        png_bytep row1 = image->row_pointers[i];
        png_bytep row2 = image->row_pointers[v_med + i - y0];
        for (int j = x0; j <= h_med; j++) {
            // Partitioning the four parts of the image into pointers
            png_bytep ptr_1 = &(row1[j * 4]);
            png_bytep ptr_2 = &(row1[(h_med + j - x0) * 4]);
            png_bytep ptr_3 = &(row2[(h_med + j - x0) * 4]);
            png_bytep ptr_4 = &(row2[j * 4]);
            if (mode == 1) {
                // Round swapping
                swap_arrays(ptr_1, ptr_2);
                swap_arrays(ptr_1, ptr_3);
                swap_arrays(ptr_1, ptr_4);
            } else {
                // Diagonal swapping
                swap_arrays(ptr_1, ptr_3);
                swap_arrays(ptr_2, ptr_4);
            }
        }
    }
}


void change_color(struct Png *image, png_color color) {
    int arr_size = (image->width) * (image->height);
    // Array to store all pixel colors png_colora
    png_colora *arr = malloc(arr_size * sizeof(png_colora));
    int c = 0;
    for (int i = 0; i < image->height; i++) {
        png_bytep row = image->row_pointers[i];
        for (int j = 0; j < image->width; j++) {
            int fl = 0;
            png_bytep ptr = &(row[j * 4]);
            // Only new colors are added to the array, and each color's
counter is incremented when repeating
            for (int k = 0; k < c; k++) {
                if ((ptr[0] == arr[k].red) && (ptr[1] == arr[k].green) &&
(ptr[2] == arr[k].blue)) {
                    arr[k].amount += 1;
                    fl = 1;
                    break;
```

```
                }
            }
            if (!fl) {
                arr[c].red = ptr[0];
                arr[c].green = ptr[1];
                arr[c].blue = ptr[2];
                c += 1;
            }
        }
    }
// Searching for a color in the array with the largest counter
    int max = 0;
    png_colora color_need_to_change;
    for (int i = 0; i < c; i++) {
        if (arr[i].amount > max) {
            max = arr[i].amount;
            color_need_to_change.red = arr[i].red;
            color_need_to_change.green = arr[i].green;
            color_need_to_change.blue = arr[i].blue;
        }
    }
// Color change
    for (int i = 0; i < image->height; i++) {
        png_bytep row = image->row_pointers[i];
        for (int j = 0; j < image->width; j++) {
            png_bytep ptr = &(row[j * 4]);
            if (ptr[0] == color_need_to_change.red && ptr[1] ==
color_need_to_change.green &&
                ptr[2] == color_need_to_change.blue) {
                ptr[0] = color.red;
                ptr[1] = color.green;
                ptr[2] = color.blue;
            }

        }
    }
}


void inversion(struct Png *image, int x0, int y0, int x1, int y1) {
    if (((x1 - x0) >= image->width) || ((y1 - y0) >= image->height)) {
        printf("the coordinates do not match the size of the image\n");
        return;
    }

    for (int i = y0; i <= y1; i++) {
        png_bytep row = image->row_pointers[i];
        for (int j = x0; j <= x1; j++) {
            png_bytep ptr = &(row[j * 4]);
            // Color inversion: 255 - <value>
            ptr[0] = 255 - ptr[0];
            ptr[1] = 255 - ptr[1];
            ptr[2] = 255 - ptr[2];
        }
    }
}
```

```
void swap_arrays(png_byte *arr1, png_byte *arr2) {
    for (int i = 0; i < 4; i++) {
        int temp = *(arr1 + i);
        *(arr1 + i) = *(arr2 + i);
        *(arr2 + i) = temp;
    }
}


bool valid_coor(int x, int y, int height, int width) {
    if (y < height && x < width && x >= 0 && y >= 0) {
        return true;
    }
    return false;
}


void set_pixel_color(struct Png *img, int x, int y, png_color color) {
    if (valid_coor(x, y, img->height, img->width) == false)
        return;
    png_bytep ptr = &(img->row_pointers[y][x * 4]);
    ptr[0] = color.red;
    ptr[1] = color.green;
    ptr[2] = color.blue;
    ptr[3] = 255;
}


void set_pixel(struct Png *des, struct Png *src, int src_x, int src_y,
int des_x, int des_y) {
    if (valid_coor(src_x, src_y, des->height, des->width) == false)
        return;
    if (valid_coor(des_x, des_y, src->height, src->width) == false)
        return;
    for (int k = 0; k < 4; k++) {
        des->row_pointers[des_y][des_x * 4 + k] =
src->row_pointers[src_y][src_x * 4 + k];
    }
}


void canvas(struct Png *image, png_byte color_type, png_byte bit_depth,
int height, int width, png_color color) {
    image->width = width;
    image->height = height;
    image->color_type = color_type;
    image->bit_depth = bit_depth;
    image->row_pointers = (png_bytep *) malloc(sizeof(png_bytep) *
height);
    for (int i = 0; i < height; i++) {
        image->row_pointers[i] = (png_byte *) malloc(sizeof(png_byte) *
width * 4);
    }
    // set all pix to white
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
```

```
            set_pixel_color(image, j, i, color);
        }
    }
}

void crop(struct Png *img, int x0, int y0, int x1, int y1) {
    struct Png copy;
    memcpy(&copy, img, sizeof(struct Png));
    png_color black = {0};
    canvas(img, img->color_type, img->bit_depth, (y1 - y0) + 1, (x1 - x0)
+ 1, black);
    for (int i = y0; i <= y1; i++) {
        for (int j = x0; j <= x1; j++) {
            set_pixel(img, &copy, j - x0, i - y0, j, i);
        }
    }
}

// paste from src to des in x,y
void paste(struct Png *des, struct Png *src, int x, int y) {
    for (int i = y; i <= y + src->height; i++) {
        for (int j = x; j <= x + src->width; j++) {
            set_pixel(des, src, j, i, j - x, i - y);
        }
    }
}
```

Название файла: functions.h

```
#include "libs_and_strucs.h"

#pragma once

void printHelp();

void swap_arrays(png_byte *arr1, png_byte *arr2);

bool valid_coor(int x, int y, int height, int width);

void set_pixel_color(struct Png *img, int x, int y, png_color color);

void set_pixel(struct Png *des, struct Png *src, int src_x, int src_y,
int des_x, int des_y);

void canvas(struct Png *image, png_byte color_type, png_byte bit_depth,
int height, int width, png_color color);

void crop(struct Png *img, int x0, int y0, int x1, int y1);

void paste(struct Png *des, struct Png *src, int x, int y);

void draw_square(struct Png *image, int x0, int y0, int size, int
line_width, png_color line_color, int is_fill,
                 png_color fill_color);

void swap_pix(struct Png *image, int mode, int x0, int y0, int x1, int
y1);
```

```
void inversion(struct Png *image, int x0, int y0, int x1, int y1);

void change_color(struct Png *image, png_color color);
```

Название файла: libs_and_strucs.h

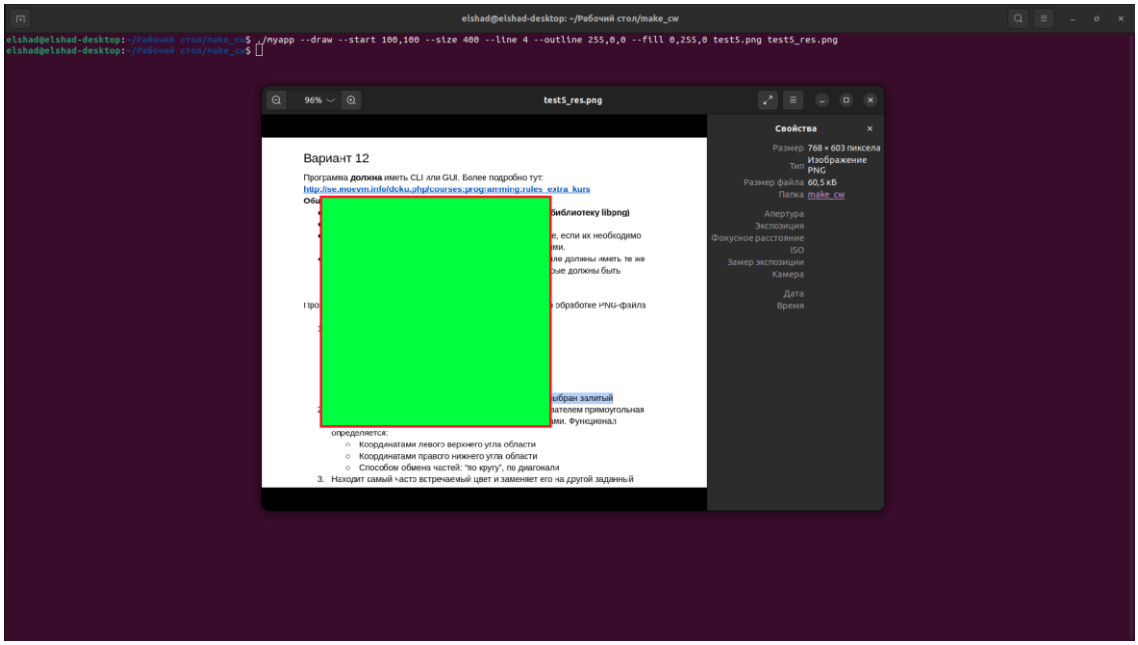
```
#include <stdlib.h>
#include <stdio.h>
#include <getopt.h>
#include <png.h>
#include <stdbool.h>
#include <unistd.h>
#include <string.h>

#pragma once

struct Png {
    int width, height;
    png_byte color_type;
    png_byte bit_depth;
    png_structp png_ptr;
    png_infop info_ptr;
    png_bytep *row_pointers;
};

typedef struct png_color_struct_amount {
    png_byte red;
    png_byte green;
    png_byte blue;
    int amount;
} png_colora;
```

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ

Рисование квадрата

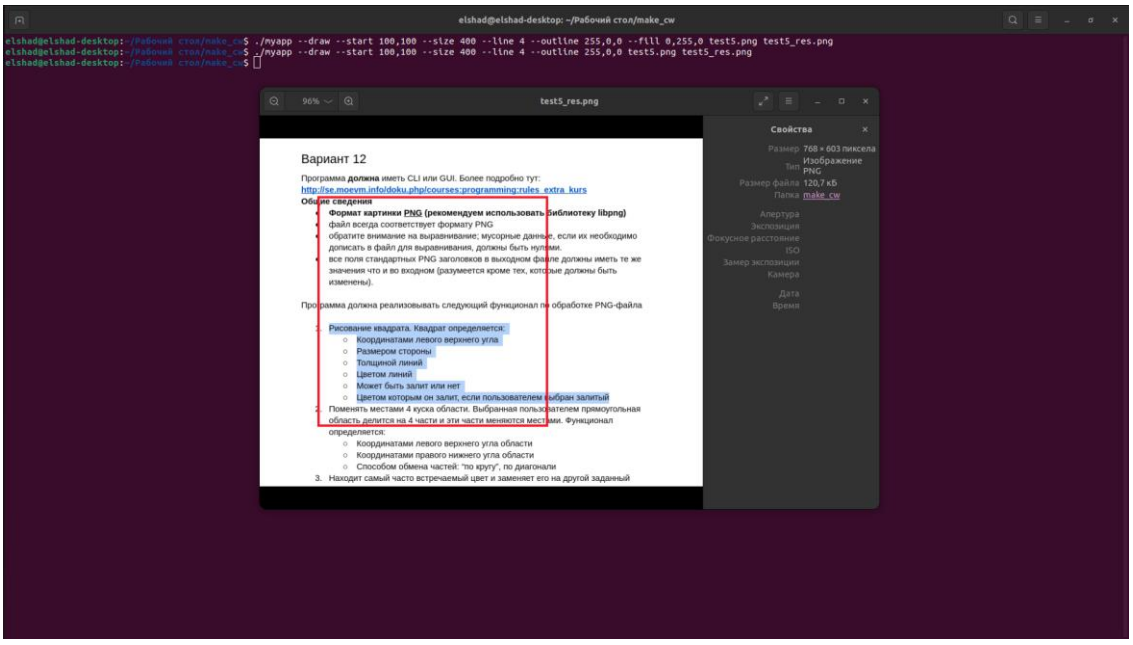
С заливкой

./myapp --draw --start 100,100 --size 400 --line 4 --outline 255,0,0 --fill 0,255,0 test5.png test5_res.png



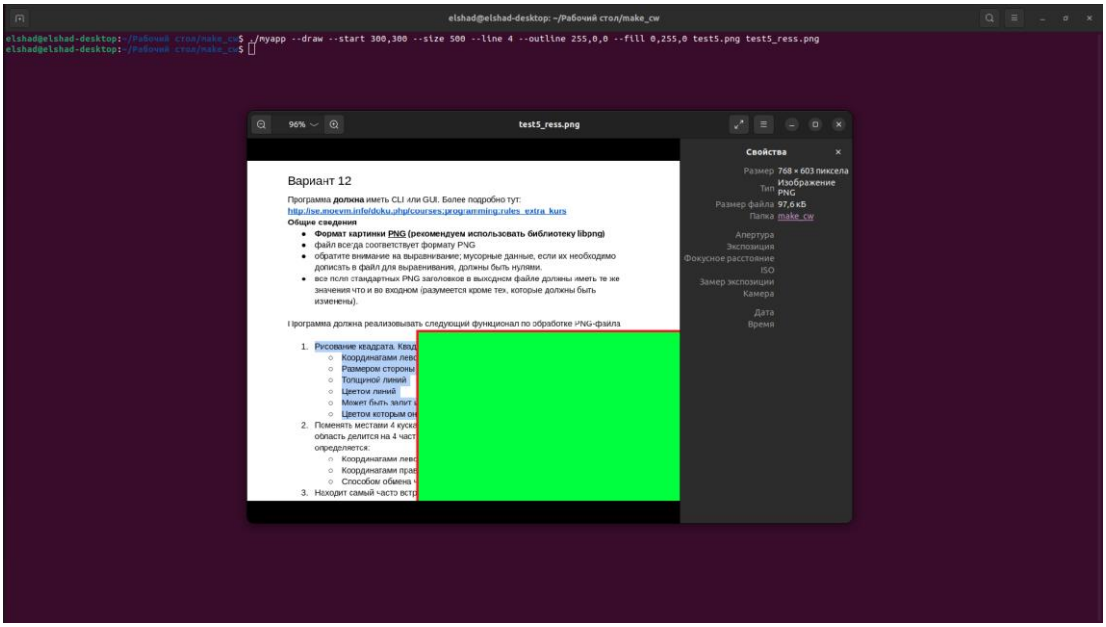
Без заливки

./myapp --draw --start 100,100 --size 400 --line 4 --outline 255,0,0 test5.png test5_res.png



Выход за границы

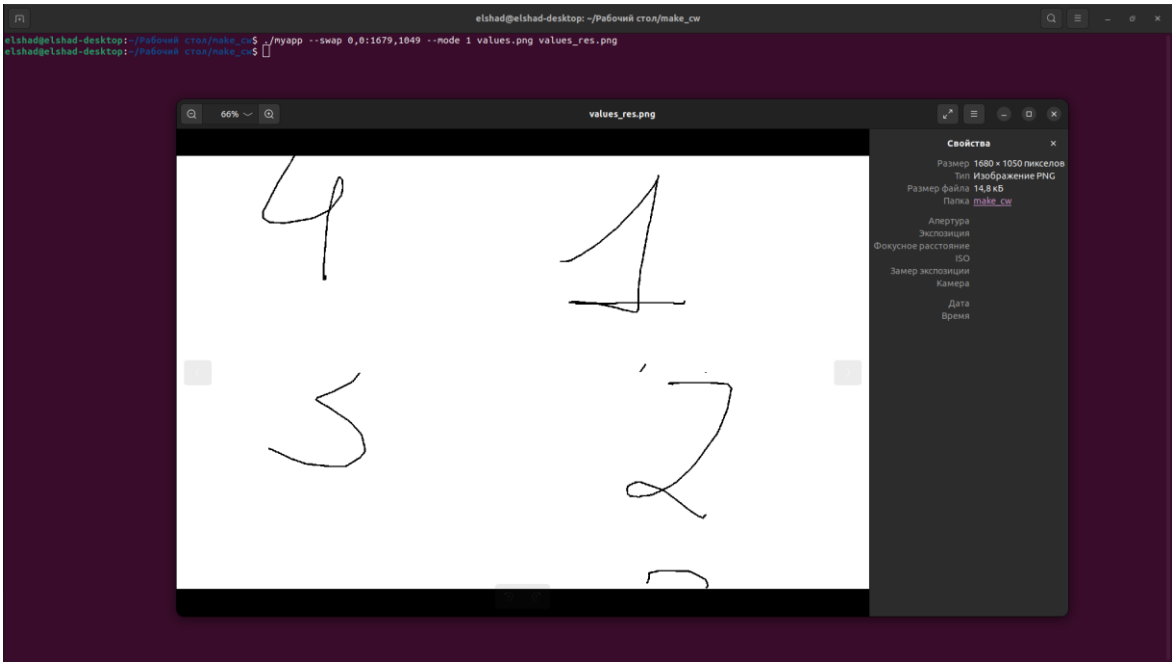
./myapp --draw --start 300,300 --size 500 --line 4 --outline 255,0,0 --fill 0,255,0 test5.png test5_ress.png



Поменять местами 4 куска области.

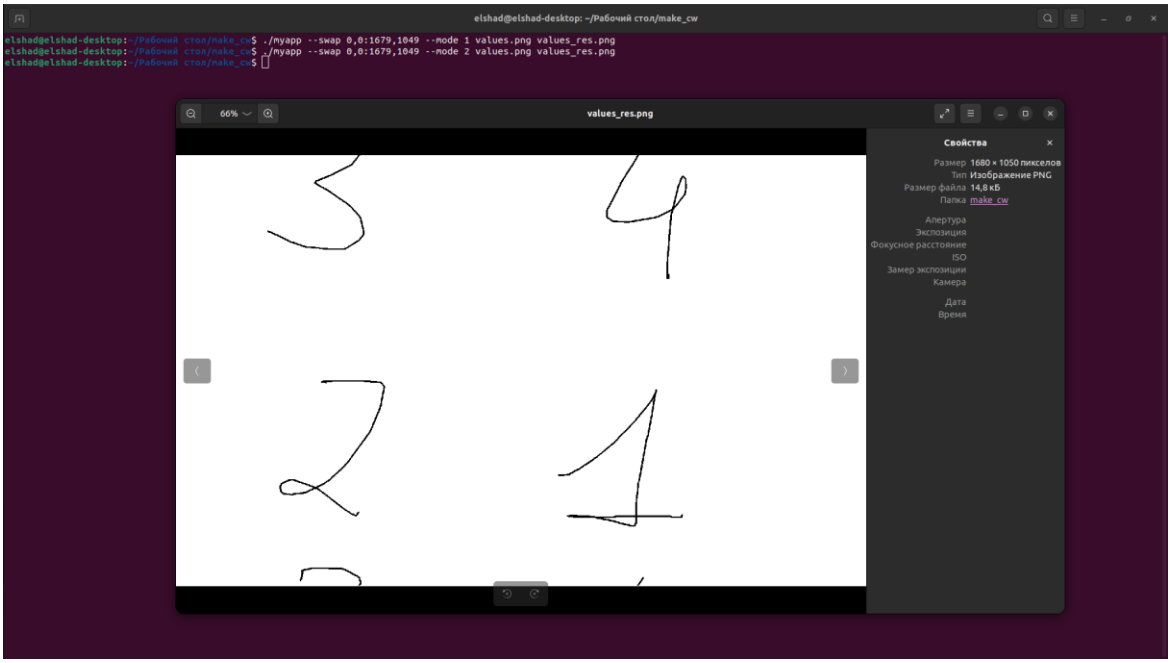
Смена по кругу

./myapp --swap 0,0:1679,1049 --mode 1 values.png values_res.png



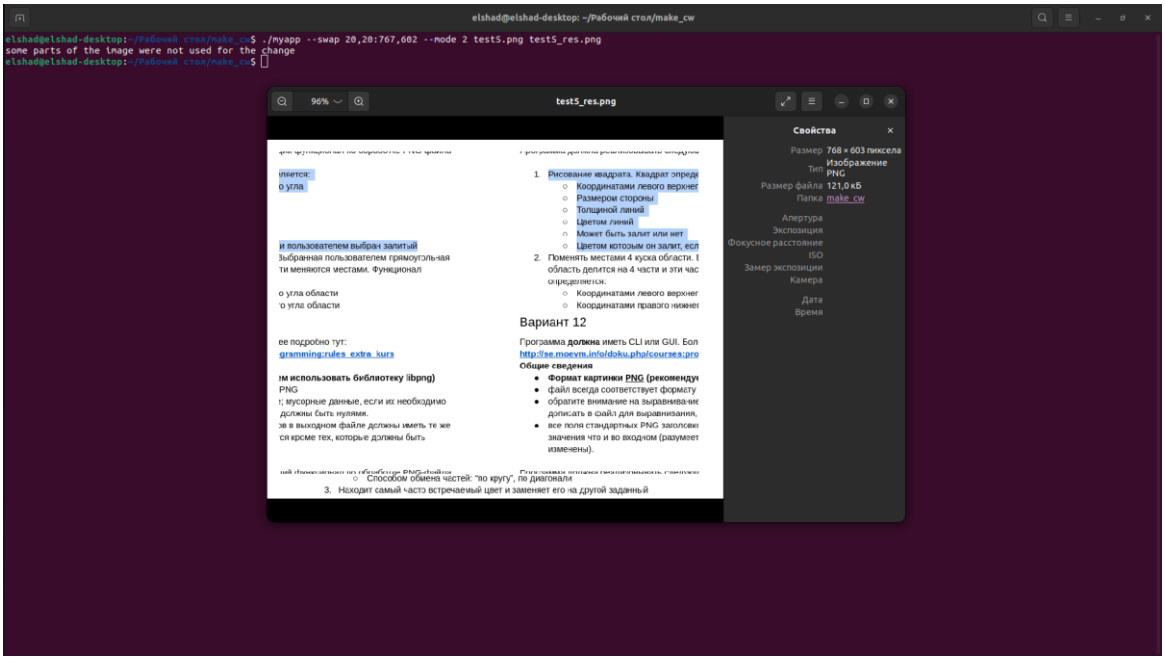
Смена по диагонали

./myapp --swap 0,0:1679,1049 --mode 2 values.png values_res.png



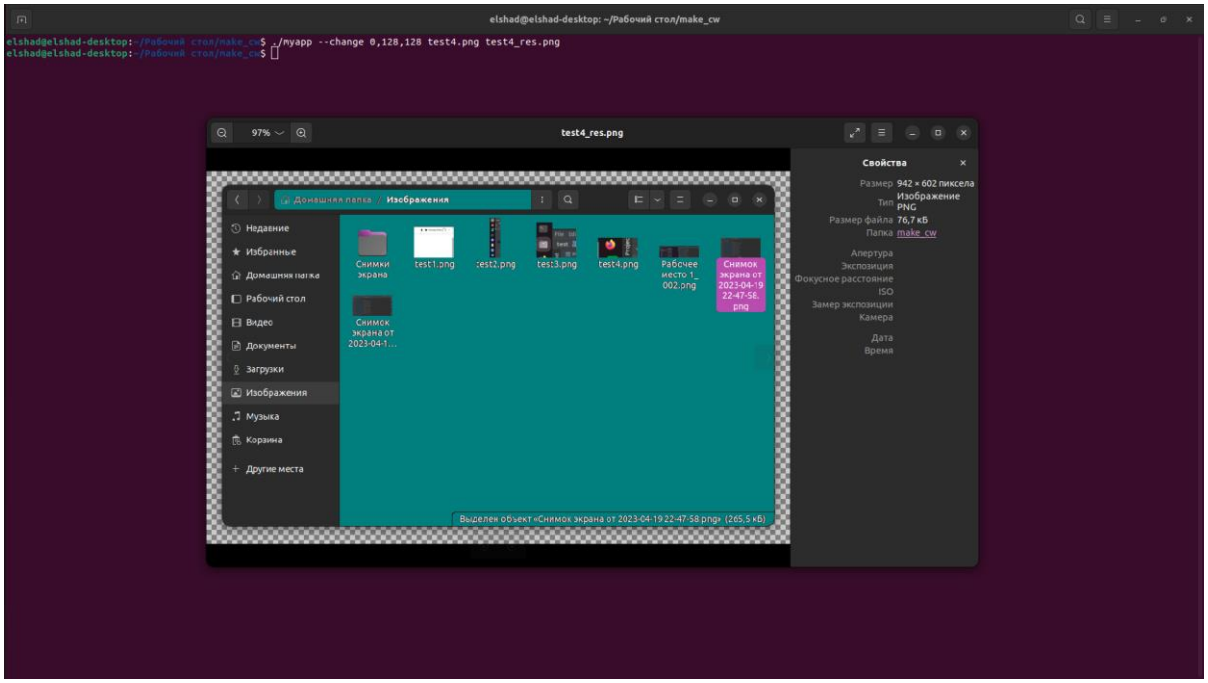
Дополнительный тест

./myapp --swap 20,20:767,602 --mode 2 test5.png test5_res.png

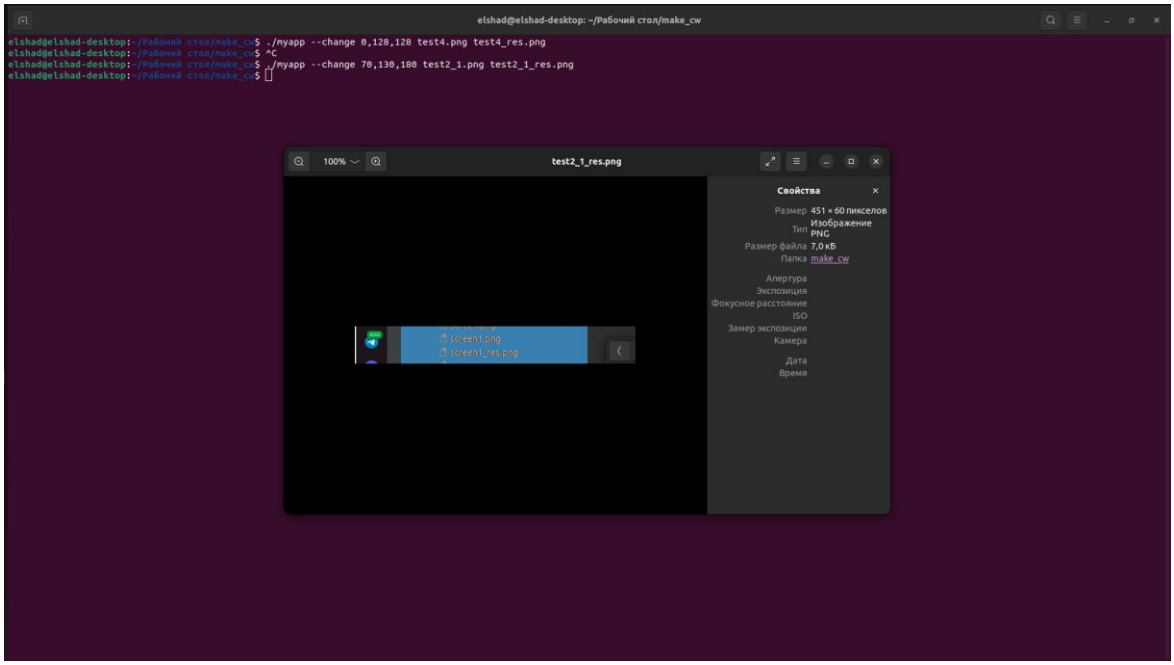


Замена цвета

./myapp --change 0,128,128 test4.png test4_res.png



./myapp --change 70,130,180 test2_1.png test2_1_res.png



Интерфейс

./myapp —help