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

**ОТЧЕТ**

**по курсовой работе**

**по дисциплине «Программирование»**

**Тема: Обработка PNG файлов.**

Студент гр. 9383 _____ Чумак М.А.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2020

**ЗАДАНИЕ**

**НА КУРСОВУЮ РАБОТУ**

Студент: Чумак М.А.

Группа: 9383

Тема работы: Обработка PNG файлов.

Исходные данные: Язык программирования C, сборка программ под Linux, файл PNG.

Содержание пояснительной записки: Введение, Задание, Содержание отчёта, Примеры работы программы, Обработка ошибок, Заключение

Предполагаемый объем пояснительной записки:
Не менее 42 страниц.

Дата выдачи задания: 02.03.2020

Дата сдачи реферата: 26.05.2020

Дата защиты реферата: 28.05.2020

Студент _____ Чумак М.А.

Преподаватель _____ Размочаева Н.В.

## Аннотация

В данной работе была разработана программа, работающая с файлами формата PNG. Для программы реализован интерфейс командной строки (CLI). Программа имеет функционал, который позволяет открывать, считывать и сохранять в файл изображение формата PNG. Выполняет такие функции по обработке PNG изображения, как рисование прямоугольника, рисование правильного шестиугольника и копирование заданной области. Также программа показывает справку для ознакомления с программой и инструкцию по правильному вводу данных.

## Оглавление

# 1. Введение

Цель работы: создание программы, которая обрабатывает файл формата PNG в зависимости от ввода пользователем различных аргументов.

Для выполнения поставленной цели были выполнены следующие задачи:

Изучить принципы работы с png файлами.
Разработать функции считывания, записи и обработки PNG файла.
Изучить принципы работы CLI.
Подробно изучить библиотеку getopt.h и разработать команды при помощи getopt_long.
Создать Makefile для сборки проекта.
Протестировать программу на различных примерах.
Найти и исправить возможные ошибки в работе программы.

# 2. Задание

Вариант 21

Программа должна иметь CLI или GUI.

Программа должна реализовывать весь следующий функционал по обработке png-файла

**Общие сведения**

- **Формат картинки PNG (рекомендуем использовать библиотеку libpng)**
- без сжатия
- 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

Рисование прямоугольника. Он определяется:

• Координатами левого верхнего угла
• Координатами правого нижнего угла
• Толщиной линий
• Цветом линий
• Прямоугольник может быть залит или нет
• Цветом которым он залит, если пользователем выбран залитый

Рисование правильного шестиугольника. Шестиугольник определяется:

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

Копирование заданной области. Функционал определяется:

• Координатами левого верхнего угла области-источника
• Координатами правого нижнего угла области-источника
• Координатами левого верхнего угла области-назначения

# 3. Содержание отчёта

## 3.1 Файлы программы

### 3.1.1 libs.h

Данный заголовочный файл содержит в себе все необходимые для работы библиотеки, объединение RGBA для хранения цветовых зарактеристик каждого пикселя, структура pixel, содержащая координаты каждого пикселя, и структура PNG для хранения всей нужной информации об изображении.

### 3.1.2 funcs.h

Данный заголовочный файл содержит объявление всех необходимых функций, которые требуются для реализации программы.

### 3.1.3 funcs.c

В данном файле описаны функции считывания, обработки и сохранения файлов формата PNG, а также функция справочной информации.

### 3.1.4 main.c

В данном файле выполняется считывание, обработка и запись файла формата PNG.

## 3.2 Описание функций

### 3.2.1 void help()

Показывает справочную информацию о программе.

### 3.2.2 void img_info(struct PNG *image)

Принимает указатель на структуру PNG, считывает и показывает информацию о файле.*png*file(char *input, struct PNG *image)

### 3.2.3 int read

Сравнивает полученную строку с именем файла, находящегося в директории, и, если удалось этот файл найти, записывает информацию в структуру image.*png*file(char *file_name, struct PNG *image)

### 3.2.4 int write

Создаёт файл формата PNG и записывает информацию из структуры image.

### 3.2.5 void clean_row(struct PNG* image)

Освобождает память под выделенные пиксели.

### 3.2.6 void pixel(struct PNG *image, int x, int y, int thickness, union RGBA * rgba1)

Записывает из структуры image в координату (x;y) пиксели в зависимости от величины толщины линии thickness цвета из объединения rgba1. Используется для рисования линий шестиугольника.

### 3.2.7 void line_hex(struct PNG *image, int x1, int y1, int x2, int y2, int thickness, union RGBA * rgba1)

Рисует линию шестиугольника от координаты (x1;y1) до (x2;y2) толщины thickness и цвета из rgba1.

### 3.2.8 void line_rect(struct PNG *image, int x, int y, union RGBA *rgba1)

Ририсует пиксель для прямоугольника, а также используется для заливки шестиугольника, в координате (x;y) цвета rgba1.

### 3.2.9 void rect(struct PNG *image, int x1, int y1, int x2, int y2, int thickness, union RGBA *rgba1)

Рисует прямоугольник от координаты (x1;y1) до (x2;y2) с толщиной линий thickness и цветом из rgba1.

### 3.2.10 void fill_rect(struct PNG *image, int x1, int y1, int x2, int y2, int thickness, union RGBA *rgba1, union RGBA *rgba2)

Рисует прямоугольник от координаты (x1;y1) до (x2;y2) с толщиной линий thickness и цветом из rgba1 и заливает его цветом из rgba2.

### 3.2.11 void hex(struct PNG *image, int x1, int y1, int x2, int y2, int thickness, union RGBA *rgba1)

Рисует шестиугольник, вписанный в окружность, которая вписана в квадрат, координаты начала которого обозначены (x1;y1), а конца (x2;y2), при помощи вызова функции line_hex 6 раз с толщиной линии thickness и цветом линий из rgba1.

### 3.2.12 void fill_hex(struct PNG *image, int x1, int y1, int x2, int y2, int thickness, union RGBA *rgba1, union RGBA *rgba2)

Рисует и заливает шестиугольник, вызывая функции hex и f_hex.

### 3.2.13 int cmp(const png_byte *ptr1, union RGBA *rgba1)

Сравнивает цвета двух полученных пикселей и в зависимости от результата выводит 0, если цвета равны, или 1, если цвета не равны. Используется для функции заливки шестиугольника.

### 3.2.14 void f_hex(struct PNG *image, int x, int y, union RGBA *rgba1, union RGBA *rgba2)

Заливает шестиугольник, начиная с центра окружности, в который вписан шестиугольник.

### 3.2.15 void copy(struct PNG *image, int x1, int y1, int x2, int y2, int x3, int y3)

Копирует область-источник с начальными (x1;y1) и конечными (x2;y2) координатами в область-назначение с начальными координатами (x3;y3).*to*int(char *a)

### 3.2.16 int char

Преобразует получаемую строку в число; в случае, если была встречена не цифра, то возвращает число -1, которое свидетельствует о неверно введённом значении.

### 3.2.17 int main(int argc, char **argv)

Функция main.c принимает аргументы int argc, которая хранит количество полученных функцией аргументов, и char **argv, которая хранит массив данных аргументов. Сначала проверяется на наличие количества аргументов, поступаемых в программу. Далее с помощью getopt_long обрабатываются аргументы при помощи ключей, описанных в справочной информации. После из файла считывается изображение, записываемое в структуру PNG image, и выполняются функции в зависимости от полученных аргументов. Также дополнительно выводится информация входного и выходного файлов. Если были получены неправильные аргументы, то будут выведены соответствующие уведомления.

# 4. Примеры работы программы

## 4.1 Рисование прямоугольника с заливкой и без

Выходной файл был получен при помощи вызова программы со следующими аргументами:

./main -i 1.png -o 12.png -R -b 100*50 -e 300*300 -t 4 -l 255*45*0_255

./main -i 12.png -o 12.png -R -b 300*200* *-e 654*354 -t 10 -l 0*2550255* *-f 0*24*354*120

Исходный файл:

Выходной файл:

## 4.2 Рисование правильного шестиугольника с заливкой и без

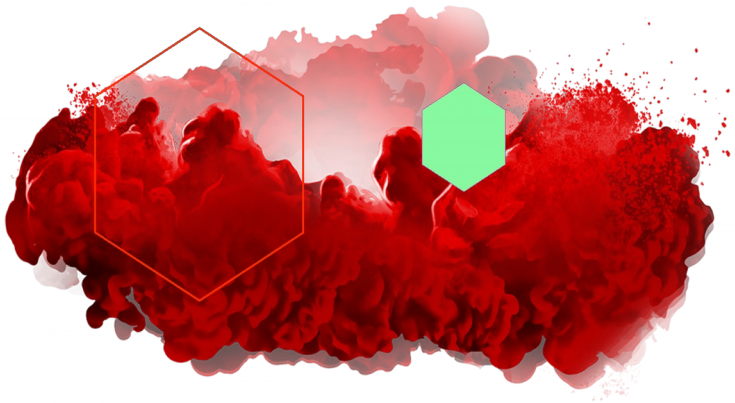
Выходной файл был получен при помощи вызова программы со следующими аргументами:

./main -i 2.png -o 22.png -H -b 200*200* -e *700*700 -t 4 -l 255*45*0_255

./main -i 22.png -o 22.png -H -r 100*1000*400 -t 1 -l 255*0*255*255* -f *0*243*54*120

Исходный файл:

Выходной файл:

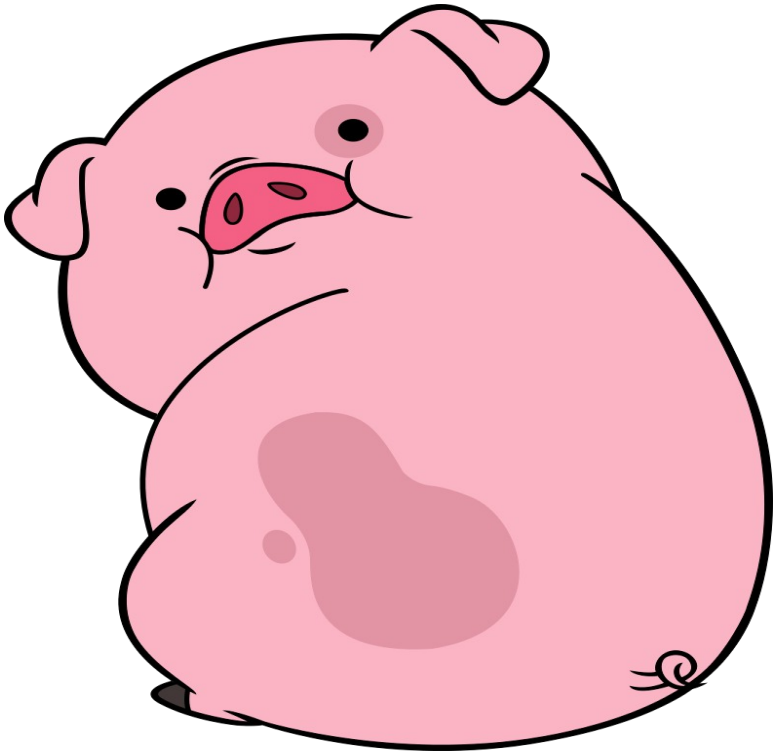


## 4.3 Копирование заданной области

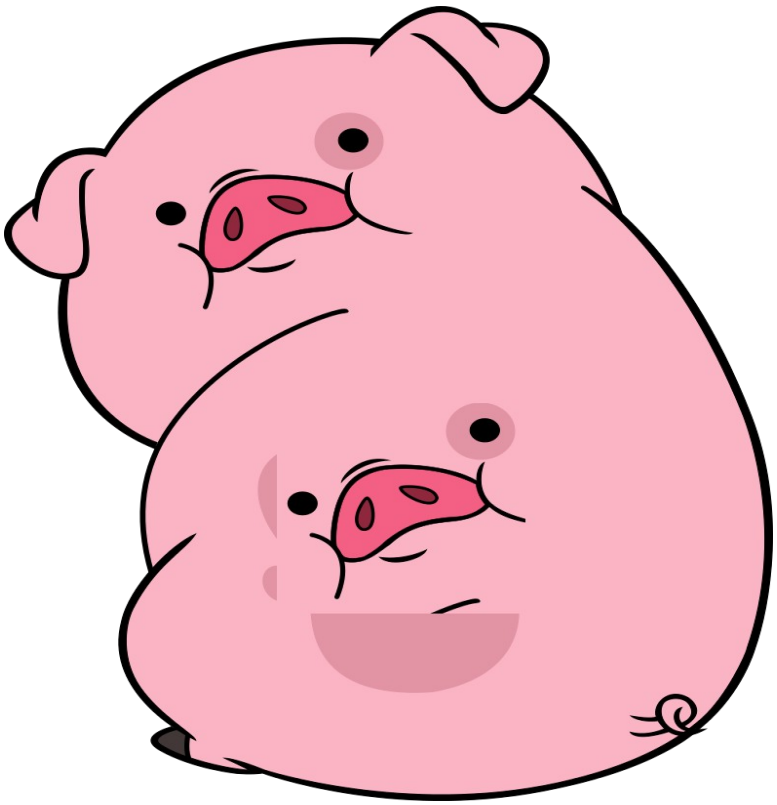
Выходной файл был получен при помощи вызова программы со следующими аргументами:

./main -i 3.png -o 32.png -C -b 145*135* -*e 400*400 -d 280_500

Исходный файл:

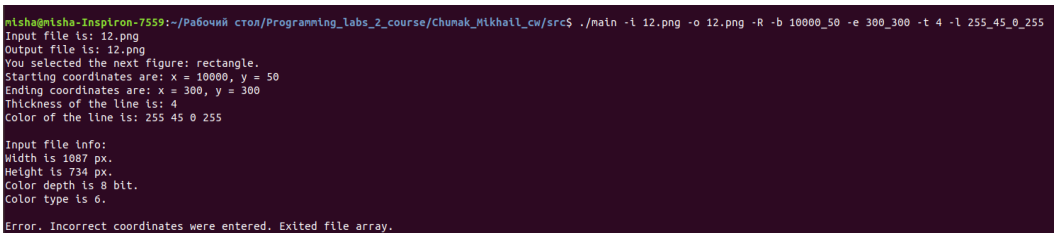


Выходной файл:

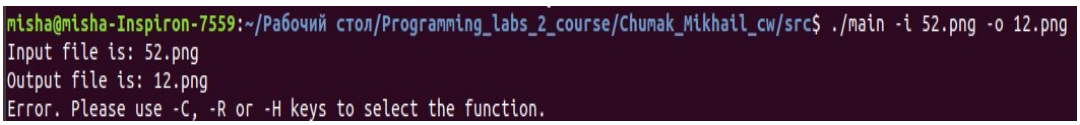


# 5. Обработка ошибок

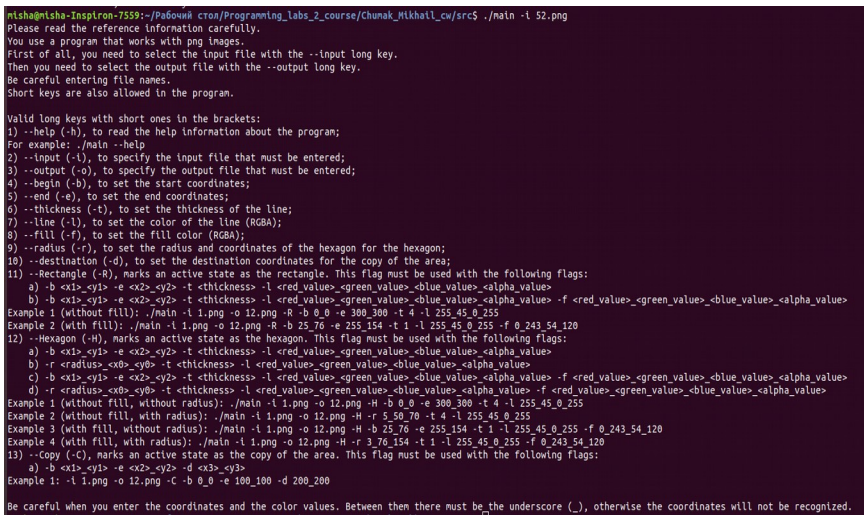
## 5.1 Ошибка при вводе неверных координат




## 5.2 Ошибка о необходимости ввести параметры функций




## 5.3 Ошибка при вводе недостаточного числа аргументов




# 6. Заключение

Изучена структура PNG файла и особенность его хранения. Созданы функции по обработке PNG изображения. Разработан CLI.

Была реализована обработка исключительных ситуаций и сделаны многочисленные тесты с рядом входных данных.

# Приложение А

## Chumak_Mikhail_cw/src/main.c

```
  1 #include "funcs.h"
  2
  3 int main(int argc, char **argv){
  4     if (argc == 1) {
  5         help();
  6         return 0;
  7     }
  8     if (argc < 5 && !((!strcmp(argv[1], "--help") || !
strcmp(argv[1], "-h")))){
  9         printf("Please read the reference information carefully.\
n");
 10         help();
 11         return 0;
 12     }
 13     if (!(!strcmp(argv[1], "--input") && !strcmp(argv[3], "--
output")) &&
 14         !(!strcmp(argv[1], "-i") && !strcmp(argv[3], "-o")) && !(!
strcmp(argv[1], "--help") || !strcmp(argv[1], "-h"))){
 15         printf("Please read the reference information carefully.\
n");
 16         help();
 17         return 0;
 18     }
 19
 20     char *short_opts = "hRHCi:o:t:b:e:l:f:r:d:";
 21     struct option long_opts[] = {
 22             {"help", no_argument,       NULL, 'h'},
 23             {"Rectangle", no_argument,       NULL, 'R'},
 24             {"Hexagon", no_argument,       NULL, 'H'},
 25             {"Copy", no_argument,       NULL, 'C'},
 26             {"input", required_argument,       NULL, 'i'},
 27             {"output", required_argument,       NULL, 'o'},
 28             {"thickness", required_argument,       NULL, 't'},
 29             {"begin", required_argument,       NULL, 'b'},
 30             {"end", required_argument,       NULL, 'e'},
 31             {"line", required_argument,       NULL, 'l'},
 32             {"fill", required_argument,       NULL, 'f'},
 33             {"radius", required_argument,       NULL, 'r'},
 34             {"destination", required_argument,       NULL, 'd'},
 35             {NULL, no_argument, NULL, 0}
 36     };
 37
 38     int opt, long_index = 0;
 39     int begin[2] = {-1, -1}, end[2] = {-1, -1}, dest[2] = {-1, -1},
 40             radius = -1, center[2] = {-1, -1},thickness = -1,  fill
= 0;
 41     union RGBA rgba1, rgba2;
 42     rgba1.ptr[0] = 0, rgba1.ptr[1] = 0, rgba1.ptr[2] = 0,
rgba1.ptr[3] = 0;
 43     rgba2.ptr[0] = 0, rgba2.ptr[1] = 0, rgba2.ptr[2] = 0,
rgba2.ptr[3] = 0;
 44     char *input = NULL, *output = NULL, *state = NULL;
 45
 46     struct PNG image;
 47     init_png(&image);
 48
 49     opt = getopt_long(argc, argv, short_opts, long_opts,
&long_index);
 50     while (opt != -1) {
 51         switch (opt) {
```

```
 52             case 'h':
 53                 help();
 54                 return 0;
 55             case 'R':
 56                 state = "rectangle";
 57                 printf("You selected the next figure: %s.\n",
state);
 58                 break;
 59             case 'H':
 60                 state = "hexagon";
 61                 printf("You selected the next figure: %s.\n",
state);
 62                 break;
 63             case 'C':
 64                 state = "copy";
 65                 printf("You selected the copy of the area.\n");
 66                 break;
 67             case 'i':
 68                 input = optarg;
 69                 printf("Input file is: %s\n", input);
 70                 break;
 71             case 'o':
 72                 output = optarg;
 73                 printf("Output file is: %s\n", output);
 74                 break;
 75             case 't':
 76                 if (!state || strcmp(state, "copy") == 0){
 77                     printf("In order to use this function, you need
to select the rectangle or the hexagon.\n");
 78                     return 0;
 79                 }
 80                 thickness = char_to_int(optarg);
 81                 if (thickness == -1){
 82                     printf("The thickness is wrong.\n");
 83                     return 0;
 84                 }
 85                 printf("Thickness of the line is: %d\n", thickness);
 86                 break;
 87             case 'b':
 88                 if (!state){
 89                     printf("In order to use this function, you need
to select the rectangle, or the hexagon, or the copy of the area.\n");
 90                     return 0;
 91                 }
 92                 begin[0] = char_to_int(strtok(optarg, "_"));
 93                 if (begin[0] == -1){
 94                     printf("The x begin coordinate is wrong.\n");
 95                     return 0;
 96                 }
 97                 begin[1] = char_to_int(strtok(NULL, "_"));
 98                 if (begin[1] == -1){
 99                     printf("The y begin coordinate is wrong.\n");
100                     return 0;
101                 }
102                 printf("Starting coordinates are: x = %d, y = %d\n",
begin[0], begin[1]);
103                 break;
104             case 'e':
105                 if (!state){
106                     printf("In order to use this function, you need
to select the rectangle, "
107                            "or the hexagon, or the copy of the
area.\n");
108                     return 0;
```

```
109                 }
110                 end[0] = char_to_int(strtok(optarg, "_"));
111                 if (end[0] == -1){
112                     printf("The x ending coordinate is wrong.\n");
113                     return 0;
114                 }
115                 end[1] = char_to_int(strtok(NULL, "_"));
116                 if (end[1] == -1){
117                     printf("The y ending coordinate is wrong.\n");
118                     return 0;
119                 }
120                 printf("Ending coordinates are: x = %d, y = %d\n",
end[0], end[1]);
121                 break;
122             case 'l':
123                 if (!state || !strcmp(state, "copy")){
124                     printf("In order to use this function, you need
to select the rectangle or the hexagon.\n");
125                     return 0;
126                 }
127                 rgba1.ptr[0] = char_to_int(strtok(optarg, "_"));
128                 rgba1.ptr[1] = char_to_int(strtok(NULL, "_"));
129                 rgba1.ptr[2] = char_to_int(strtok(NULL, "_"));
130                 rgba1.ptr[3] = char_to_int(strtok(NULL, "_"));
131                 printf("Color of the line is: %d %d %d %d\n",
rgba1.ptr[0], rgba1.ptr[1], rgba1.ptr[2], rgba1.ptr[3]);
132                 break;
133             case 'f':
134                 if (!state || !strcmp(state, "copy")){
135                     printf("In order to use this function, you need
to select the rectangle or the hexagon.\n");
136                     return 0;
137                 }
138                 fill = 1;
139                 rgba2.ptr[0] = char_to_int(strtok(optarg, "_"));
140                 rgba2.ptr[1] = char_to_int(strtok(NULL, "_"));
141                 rgba2.ptr[2] = char_to_int(strtok(NULL, "_"));
142                 rgba2.ptr[3] = char_to_int(strtok(NULL, "_"));
143                 printf("Filled color is: %d %d %d %d\n",
rgba2.ptr[0], rgba2.ptr[1], rgba2.ptr[2], rgba2.ptr[3]);
144                 break;
145             case 'r':
146                 if (!state || !strcmp(state, "copy")){
147                     printf("In order to use this function, you need
to select the rectangle or the hexagon.\n");
148                     return 0;
149                 }
150                 radius = char_to_int(strtok(optarg, "_"));
151                 if (radius == -1){
152                     printf("The radius is wrong.\n");
153                     return 0;
154                 }
155                 center[0] = char_to_int(strtok(NULL, "_"));
156                 if (center[0] == -1){
157                     printf("The x center coordinate is wrong.\n");
158                     return 0;
159                 }
160                 center[1] = char_to_int(strtok(NULL, "_"));
161                 if (center[1] == -1){
162                     printf("The y center coordinate is wrong.\n");
163                     return 0;
164                 }
165                 printf("Radius and center coordinates are: %d %d %d\
n", radius, center[0], center[1]);
```

```
166                 break;
167             case 'd':
168                 if (!state || !strcmp(state, "rectangle") || !
strcmp(state, "hexagon")){
169                     printf("In order to use this function, you need
to select the copy of the area.\n");
170                     return 0;
171                 }
172                 if (begin[0] == -1 || begin[1] == -1 || end[0] == -1
|| end[1] == -1){
173                     printf("In order to use this function, you need
to enter the start and the end coordinates.\n");
174                     return 0;
175                 }
176                 dest[0] = char_to_int(strtok(optarg, "_"));
177                 if (dest[0] == -1){
178                     printf("The x destination coordinate is wrong.\
n");
179                     return 0;
180                 }
181                 dest[1] = char_to_int(strtok(NULL, "_"));
182                 if (dest[1] == -1){
183                     printf("The y destination coordinate is wrong.\
n");
184                     return 0;
185                 }
186                 printf("Destination coordinates are: x = %d, y = %d\
n", dest[0], dest[1]);
187                 break;
188             default:
189                 break;
190         }
191         opt = getopt_long(argc, argv, short_opts, long_opts,
&long_index);
192     }
193     if(!state){
194         printf("Error. Please use -C, -R or -H keys to select the
function.\n");
195         return 0;
196     }
197     if ((begin[0] < 0 || begin[1] < 0 || end[0] < 0 || end[1] < 0)
&& !strcmp(state, "rectangle") && !strcmp(state, "copy")){
198         printf("Error. Incorrect coordinates were entered. The
coordinates "
199                " must not be <0 value.\n");
200         return 0;
201     }
202     if (((begin[0] < 0 || begin[1] < 0 || end[0] < 0 || end[1] < 0))
&& !strcmp(state, "hexagon") && radius <= 0){
203         printf("Error. Incorrect coordinates were entered. The
coordinates "
204                "must not be <0 value.\n");
205         return 0;
206     }
207     if (thickness <= 0 && strcmp(state, "copy") != 0){
208         printf("Error. The thickness of the line must not be <=0
value.\n");
209         return 0;
210     }
211     if (!strcmp(state, "copy") && (dest[0] < 0 || dest[1] < 0)){
212         printf("Error. Please enter the correct value of the
destination coordinates.\n");
213         return 0;
214     }
```

```
215     if ((begin[0] > end[0] || begin[1] > end[1]) && !radius){
216         printf("Error. Incorrect coordinates were entered. The
starting coordinates must be less "
217                "than the ending coordinates. Maybe there are radius
with 0 value.\n");
218         return 0;
219     }
220     if ((begin[0] == end[0] || begin[1] == end[1]) && radius <= 0){
221         printf("Error. Incorrect coordinates were entered. The
starting "
222                "coordinates must not be equal to the ending ones.\
n");
223         return 0;
224     }
225
226     if (read_png_file(input, &image))
227         return 0;
228     if (png_get_color_type(image.png_ptr, image.info_ptr) ==
PNG_COLOR_TYPE_RGB){
229         printf("This program only works with PNG files of the RGBA
type.\n");
230         clean_row(&image);
231         return 0;
232     }
233
234     printf("\nInput file info:\n");
235     img_info(&image);
236
237     if (!strcmp(state, "copy") && ((end[0] >= image.width) ||
(end[1] >= image.height) ||
238                                    (dest[0] + (end[0] - begin[0]) >=
image.width) || (dest[1] + (end[1] - begin[1])  >= image.height))){
239         printf("Error. Please enter the correct value of the
coordinates.\n");
240         return 0;
241     }
242     if ((thickness/2 > abs(image.height - end[1]) && end[1] >= 0 ||
thickness/2 > begin[1] && begin[1] >= 0
243          || thickness/2 > abs(image.width - end[0]) && end[0] >= 0
|| thickness/2 >= begin[0] && begin[0] > 0) && !strcmp(state,
"hexagon")){
244         printf("Error. The frames are too big. The width of the
frames "
245                "should be a maximum of half the height of the
figure.\n");
246         clean_row(&image);
247         return 0;
248     }
249     if (((thickness/2 > abs(image.height - center[1]) && center[1] >
0 ||
250           thickness/2 > abs(image.width - center[0]) && center[0] >
0)
251          || ((thickness/2 > center[0]-radius) && center[0] > 0))
&& !strcmp(state, "hexagon")){
252         printf("Error. The frames are too big. The width of the
frames "
253                "should be a maximum of half the height of the
figure.\n");
254         clean_row(&image);
255         return 0;
256     }
257     if ((center[0] + radius >= image.width) || (center[1] + radius
>= image.height)){
258         printf("Error. The frames are too big.\n");
```

```
259         clean_row(&image);
260         return 0;
261     }
262     if (begin[0] >= image.width || begin[1] >= image.height ||
end[0] >= image.width || end[1] >= image.height){
263         printf("Error. Incorrect coordinates were entered. Exited
file array.\n");
264         clean_row(&image);
265         return 0;
266     }
267 
268     if (!strcmp(state, "rectangle") && fill == 1){
269         state = "filledRect";
270     }
271     if (!strcmp(state, "hexagon") && fill == 1){
272         state = "filledHex";
273     }
274 
275     if (!strcmp(state, "rectangle")){
276         rect(&image, begin[0], begin[1], end[0], end[1], thickness,
&rgba1);
277     } else if (!strcmp(state, "filledRect")){
278         fill_rect(&image, begin[0], begin[1], end[0], end[1],
thickness, &rgba1, &rgba2);
279     } else if (!strcmp(state, "hexagon")){
280         if (center[0] > 0 && center[1] > 0){
281             begin[0] = center[0];
282             begin[1] = center[1];
283         }
284         if (radius > 0){
285             end[0] = begin[0] + radius;
286             end[1] = begin[1] + radius;
287             begin[0] = begin[0] - radius;
288             begin[1] = begin[1] - radius;
289         } else
290         if (end[0] - begin[0] != end[1] - begin[1]){
291             printf("Error. It is not a square.\n");
292             clean_row(&image);
293             return 0;
294         }
295         if (begin[0] < 0 || end[0] < 0 || begin[1] < 0 || end[1] <
0){
296             printf("Error. Incorrect coordinates were entered.
Exited file array.\n");
297             clean_row(&image);
298             return 0;
299         }
300         hex(&image, begin[0], begin[1], end[0], end[1], thickness,
&rgba1);
301     } else if (!strcmp(state, "filledHex")){
302         if (center[0] > 0 && center[1] > 0){
303             begin[0] = center[0];
304             begin[1] = center[1];
305         }
306         if (radius > 0){
307             end[0] = begin[0] + radius;
308             end[1] = begin[1] + radius;
309             begin[0] = begin[0] - radius;
310             begin[1] = begin[1] - radius;
311         } else
312         if (end[0] - begin[0] != end[1] - begin[1]){
313             printf("Error. It is not a square.\n");
314             clean_row(&image);
315             return 0;
```

```
316         }
317         if (begin[0] < 0 || end[0] < 0 || begin[1] < 0 || end[1] <
0){
318             printf("Error. Incorrect coordinates were entered.
Exited file array.\n");
319             clean_row(&image);
320             return 0;
321         }
322         fill_hex(&image, begin[0], begin[1], end[0], end[1],
thickness, &rgba1, &rgba2);
323     } else if (!strcmp(state, "copy"))
324         copy(&image, begin[0], begin[1], end[0], end[1], dest[0],
dest[1]);
325
326     if (write_png_file(output, &image)){
327         clean_row(&image);
328         return 0;
329     }
330     printf("Output file info:\n");
331     img_info(&image);
332
333     return 0;
334 }
```

## Chumak_Mikhail_cw/src/funcs.c

```
  1 #include "funcs.h"
  2
  3 void help(){
  4     printf("You use a program that works with png images.\n"
  5            "First of all, you need to select the input file with the
--input long key.\n"
  6            "Then you need to select the output file with the --
output long key.\n"
  7            "Be careful entering file names.\n"
  8            "Short keys are also allowed in the program.\n\n"
  9            "Valid long keys with short ones in the brackets:\n"
 10            "1) --help (-h), to read the help information about the
program;\n"
 11            "For example: ./main --help\n"
 12            "2) --input (-i), to specify the input file that must be
entered;\n"
 13            "3) --output (-o), to specify the output file that must
be entered;\n"
 14            "4) --begin (-b), to set the start coordinates;\n"
 15            "5) --end (-e), to set the end coordinates;\n"
 16            "6) --thickness (-t), to set the thickness of the line;\
n"
 17            "7) --line (-l), to set the color of the line (RGBA);\n"
 18            "8) --fill (-f), to set the fill color (RGBA);\n"
 19            "9) --radius (-r), to set the radius and coordinates of
the hexagon for the hexagon;\n"
 20            "10) --destination (-d), to set the destination
coordinates for the copy of the area;\n"
 21            "11) --Rectangle (-R), marks an active state as the
rectangle. This flag must be used with the following flags:\n"
 22            "    a) -b <x1>_<y1> -e <x2>_<y2> -t <thickness> -l
<red_value>_<green_value>_<blue_value>_<alpha_value>\n"
 23            "    b) -b <x1>_<y1> -e <x2>_<y2> -t <thickness> -l
<red_value>_<green_value>_<blue_value>_<alpha_value> -f
<red_value>_<green_value>_<blue_value>_<alpha_value>\n"
 24            "Example 1 (without fill): ./main -i 1.png -o 12.png -R -
b 0_0 -e 300_300 -t 4 -l 255_45_0_255\n"
 25            "Example 2 (with fill): ./main -i 1.png -o 12.png -R -b
25_76 -e 255_154 -t 1 -l 255_45_0_255 -f 0_243_54_120\n"
```

```
 26            "12) --Hexagon (-H), marks an active state as the
hexagon. This flag must be used with the following flags:\n"
 27            "    a) -b <x1>_<y1> -e <x2>_<y2> -t <thickness> -l
<red_value>_<green_value>_<blue_value>_<alpha_value>\n"
 28            "    b) -r <radius>_<x0>_<y0> -t <thickness> -l
<red_value>_<green_value>_<blue_value>_<alpha_value>\n"
 29            "    c) -b <x1>_<y1> -e <x2>_<y2> -t <thickness> -l
<red_value>_<green_value>_<blue_value>_<alpha_value> -f
<red_value>_<green_value>_<blue_value>_<alpha_value>\n"
 30            "    d) -r <radius>_<x0>_<y0> -t <thickness> -l
<red_value>_<green_value>_<blue_value>_<alpha_value> -f
<red_value>_<green_value>_<blue_value>_<alpha_value>\n"
 31            "Example 1 (without fill, without radius): ./main -i
1.png -o 12.png -H -b 0_0 -e 300_300 -t 4 -l 255_45_0_255\n"
 32            "Example 2 (without fill, with radius): ./main -i 1.png -
o 12.png -H -r 5_50_70 -t 4 -l 255_45_0_255\n"
 33            "Example 3 (with fill, without radius): ./main -i 1.png -
o 12.png -H -b 25_76 -e 255_154 -t 1 -l 255_45_0_255 -f 0_243_54_120\n"
 34            "Example 4 (with fill, with radius): ./main -i 1.png -o
12.png -H -r 3_76_154 -t 1 -l 255_45_0_255 -f 0_243_54_120\n"
 35            "13) --Copy (-C), marks an active state as the copy of
the area. This flag must be used with the following flags:\n"
 36            "    a) -b <x1>_<y1> -e <x2>_<y2> -d <x3>_<y3>\n"
 37            "Example 1: -i 1.png -o 12.png -C -b 0_0 -e 100_100 -d
200_200\n\n"
 38            "Be careful when you enter the coordinates and the color
values. Between them there must be the underscore (_), otherwise the
coordinates will not be recognized.\n");
 39 }
 40
 41 void init_png(struct PNG *image){
 42     image->height = 0;
 43     image->width = 0;
 44     image->png_ptr = NULL;
 45     image->bit_depth = 0;
 46     image->color_type = 0;
 47     image->info_ptr = NULL;
 48     image->number_of_passes = 0;
 49     image->row_pointers = NULL;
 50 }
 51
 52 void img_info(struct PNG *image){
 53     printf("Width is %d px.\nHeight is %d px.\nColor depth is %d
bit.\nColor type is %d.\n\n",
 54            image->width, image->height, image->bit_depth, image-
>color_type);
 55 }
 56
 57 int read_png_file(char *input, struct PNG *image){
 58     int y;
 59     png_byte header[8];
 60
 61     FILE *fp = fopen(input, "rb");
 62     if (!fp){
 63         printf("Error. The input file is not found.\n");
 64         return 1;
 65     }
 66
 67     fread(header, 1, 8, fp);
 68     if (png_sig_cmp(header, 0, 8)) {
 69         printf("Error. The file is not a PNG image!\n");
 70         fclose(fp);
 71         return 1;
 72     }
```

```
 73
 74     image->png_ptr = png_create_read_struct(PNG_LIBPNG_VER_STRING,
NULL, NULL, NULL);
 75
 76     if (!image->png_ptr){
 77         png_destroy_read_struct(&image->png_ptr, &image->info_ptr,
NULL);
 78         printf("The PNG file is invalid.\n");
 79         fclose(fp);
 80         return 1;
 81     }
 82
 83     image->info_ptr = png_create_info_struct(image->png_ptr);
 84     if (!image->info_ptr){
 85         png_destroy_read_struct(&image->png_ptr, &image->info_ptr,
NULL);
 86         printf("The PNG file is invalid.\n");
 87         fclose(fp);
 88         return 1;
 89     }
 90
 91     if (setjmp(png_jmpbuf(image->png_ptr))){
 92         png_destroy_read_struct(&image->png_ptr, &image->info_ptr,
NULL);
 93         printf("The PNG file is invalid.\n");
 94         fclose(fp);
 95         return 1;
 96     }
 97
 98     png_init_io(image->png_ptr, fp);
 99     png_set_sig_bytes(image->png_ptr, 8);
100
101     png_read_info(image->png_ptr, image->info_ptr);
102
103     image->width = png_get_image_width(image->png_ptr, image-
>info_ptr);
104     image->height = png_get_image_height(image->png_ptr, image-
>info_ptr);
105     image->color_type = png_get_color_type(image->png_ptr, image-
>info_ptr);
106     image->bit_depth = png_get_bit_depth(image->png_ptr, image-
>info_ptr);
107
108     image->number_of_passes = png_set_interlace_handling(image-
>png_ptr);
109     png_read_update_info(image->png_ptr, image->info_ptr);
110
111     if (setjmp(png_jmpbuf(image->png_ptr))){
112         png_destroy_read_struct(&image->png_ptr, &image->info_ptr,
NULL);
113         printf("The PNG file is invalid.\n");
114         fclose(fp);
115         return 1;
116     }
117
118     image->row_pointers = (png_bytep *) malloc(sizeof(png_bytep) *
image->height);
119     for (y = 0; y < image->height; y++)
120         image->row_pointers[y] = (png_byte *)
malloc(png_get_rowbytes(image->png_ptr, image->info_ptr));
121
122     png_read_image(image->png_ptr, image->row_pointers);
123
124     fclose(fp);
```

```
125     return 0;
126 }
127
128 int write_png_file(char *file_name, struct PNG *image){
129     int y;
130     FILE *fp = fopen(file_name, "wb");
131     if (!fp){
132         printf("Error writing file.\n");
133         fclose(fp);
134         return 1;
135     }
136
137     image->png_ptr = png_create_write_struct(PNG_LIBPNG_VER_STRING,
NULL, NULL, NULL);
138
139     if (!image->png_ptr){
140         png_destroy_write_struct(&image->png_ptr, &image->info_ptr);
141         printf("Error writing file.");
142         fclose(fp);
143         return 1;
144     }
145
146     image->info_ptr = png_create_info_struct(image->png_ptr);
147     if (!image->info_ptr){
148         png_destroy_write_struct(&image->png_ptr, &image->info_ptr);
149         printf("Error writing file.");
150         fclose(fp);
151         return 1;
152     }
153
154     if (setjmp(png_jmpbuf(image->png_ptr))){
155         png_destroy_write_struct(&image->png_ptr, &image->info_ptr);
156         printf("Error writing file.");
157         fclose(fp);
158         return 1;
159     }
160
161     png_init_io(image->png_ptr, fp);
162
163     if (setjmp(png_jmpbuf(image->png_ptr))){
164         png_destroy_write_struct(&image->png_ptr, &image->info_ptr);
165         printf("Error writing file.");
166         fclose(fp);
167         return 1;
168     }
169
170     png_set_IHDR(image->png_ptr, image->info_ptr, image->width,
image->height,
171                  image->bit_depth, image->color_type,
PNG_INTERLACE_NONE,
172                  PNG_COMPRESSION_TYPE_BASE, PNG_FILTER_TYPE_BASE);
173
174     png_write_info(image->png_ptr, image->info_ptr);
175
176     if (setjmp(png_jmpbuf(image->png_ptr))){
177         png_destroy_write_struct(&image->png_ptr, &image->info_ptr);
178         printf("Error writing file.");
179         fclose(fp);
180         return 1;
181     }
182
183     png_write_image(image->png_ptr, image->row_pointers);
184
185     if (setjmp(png_jmpbuf(image->png_ptr))){
```

```
186         png_destroy_write_struct(&image->png_ptr, &image->info_ptr);
187         printf("Error writing file.");
188         fclose(fp);
189         return 1;
190     }
191 
192     png_write_end(image->png_ptr, NULL);
193 
194     for (y = 0; y < image->height; y++)
195         free(image->row_pointers[y]);
196     free(image->row_pointers);
197 
198     fclose(fp);
199 
200     return 0;
201 }
202 
203 void clean_row(struct PNG* image){
204     for (int i = 0; i < image->height; i++)
205         free(image->row_pointers[i]);
206     free(image->row_pointers);
207 }
208 
209 void pixel(struct PNG *image, int x, int y, int thickness, union
RGBA * rgba1){
210     for(int i = -thickness/2; i < thickness - thickness/2; i++)
211         for(int j = -thickness/2; j < thickness - thickness/2; j++){
212             if(y+i >= 0 && y+i < image->height && x+j >= 0 && x+j <
image->width && i*i+j*j<=thickness*thickness/4){
213                 png_byte *ptr = &(image->row_pointers[y+i]
[(x+j)*4]);
214                 ptr[0] = rgba1->ptr[0];
215                 ptr[1] = rgba1->ptr[1];
216                 ptr[2] = rgba1->ptr[2];
217                 ptr[3] = rgba1->ptr[3];
218             }
219         }
220 }
221 
222 void line_vd_hex(struct PNG *image, int x, int y, int length, int
lengthX, int lengthY, int dx, int dy, int thickness, union RGBA * rgba1,
int k){
223     int d = -lengthX;
224     length++;
225     while(length--){
226         if (k == 0)
227             pixel(image, x, y, thickness, rgba1);
228         else
229             pixel(image, y, x, thickness, rgba1);
230         x += dx;
231         d += 2 * lengthY;
232         if (d > 0){
233             d -= 2 * lengthX;
234             y += dy;
235         }
236     }
237 }
238 
239 void line_hex(struct PNG *image, int x1, int y1, int x2, int y2, int
thickness, union RGBA * rgba1){
240     int dx = (x2 - x1 >= 0 ? 1 : -1);
241     int dy = (y2 - y1 >= 0 ? 1 : -1);
242     int lengthX = abs(x2 - x1);
243     int lengthY = abs(y2 - y1);
```

```
244     int length;
245     int x = x1, y = y1, k = 0;
246     if (lengthX > lengthY)
247         length = lengthX;
248     else
249         length = lengthY;
250     if (length == 0)
251         pixel(image, x1, y1, thickness, rgba1);
252     if (lengthY <= lengthX)
253         line_vd_hex(image, x, y, length, lengthX, lengthY, dx, dy,
thickness, rgba1, k);
254     else {
255         k = 1;
256         line_vd_hex(image, y, x, length, lengthY, lengthX, dy, dx,
thickness, rgba1, k);
257     }
258 }
259
260 void line_rect(struct PNG *image, int x, int y, union RGBA *rgba1){
261     png_byte *ptr = &(image->row_pointers[y][x * 4]);
262     ptr[0] = rgba1->ptr[0];
263     ptr[1] = rgba1->ptr[1];
264     ptr[2] = rgba1->ptr[2];
265     ptr[3] = rgba1->ptr[3];
266 }
267
268 void rect(struct PNG *image, int x1, int y1, int x2, int y2, int
thickness, union RGBA *rgba1) {
269     for (int y = y1; y <= y2; y++)
270         for (int x = x1; x <= x2; x++){
271             if (x < x1 + thickness || x > x2 - thickness || y < y1 +
thickness || y > y2 - thickness){
272                 line_rect(image, x, y, rgba1);
273             }
274         }
275 }
276
277 void fill_rect(struct PNG *image, int x1, int y1, int x2, int y2,
int thickness, union RGBA *rgba1, union RGBA *rgba2){
278     for (int y = y1; y <= y2; y++)
279         for (int x = x1; x <= x2; x++){
280             if (x < x1 + thickness || x > x2 - thickness || y < y1 +
thickness || y > y2 - thickness){
281                 line_rect(image, x, y, rgba1);
282             }
283         }
284     for (int y = y1 + thickness; y < y2 - thickness + 1; y++) {
285         for (int x = x1 + thickness; x < x2 - thickness + 1; x++) {
286             line_rect(image, x, y, rgba2);
287         }
288     }
289 }
290
291 void hex(struct PNG *image, int x1, int y1, int x2, int y2, int
thickness, union RGBA *rgba1){
292     int center[2] = {(x2+x1)/2, (y2+y1)/2};
293     int radius = (x2-x1)/2;
294     int arr[6][2];
295     arr[0][0] = center[0]; arr[0][1] = center[1]-radius;
296     arr[1][0] = (int)(center[0]-radius*cos(M_PI/6)); arr[1][1] =
center[1]-radius/2;
297     arr[2][0] = (int)(center[0]-radius*cos(M_PI/6)); arr[2][1] =
center[1]+radius/2;
298     arr[3][0] = center[0]; arr[3][1] = center[1]+radius;
```

```
299     arr[4][0] = (int)(center[0]+radius*cos(M_PI/6)); arr[4][1] =
center[1]+radius/2;
300     arr[5][0] = (int)(center[0]+radius*cos(M_PI/6)); arr[5][1] =
center[1]-radius/2;
301     for (int i = 0, j = 1; i < 6; i++, j++){
302         if (i == 5)
303             j = 0;
304         line_hex(image, arr[i][0], arr[i][1], arr[j][0], arr[j][1],
thickness, rgba1);
305     }
306 }
307
308 void fill_hex(struct PNG *image, int x1, int y1, int x2, int y2, int
thickness, union RGBA *rgba1, union RGBA *rgba2){
309     hex(image, x1, y1, x2, y2, thickness, rgba1);
310     f_hex(image, x2 - (x2 - x1) / 2, y2 - (y2 - y1) / 2, rgba1,
rgba2);
311 }
312
313 int cmp(const png_byte *ptr1, union RGBA *rgba1){
314     if (ptr1[0] == rgba1->ptr[0] && ptr1[1] == rgba1->ptr[1] &&
ptr1[2] == rgba1->ptr[2] &&
315         ptr1[3] == rgba1->ptr[3])
316         return 0;
317     else return 1;
318 }
319
320 void fill_checker(struct PNG *image, struct pixel **ptrbuf, int *i,
png_byte *ptr2, int x1, int y1, union RGBA *rgba1, union RGBA *rgba2){
321     ptr2 = &(image->row_pointers[y1][x1 * 4]);
322     if (cmp(ptr2, rgba1) && cmp(ptr2, rgba2)){
323         *ptrbuf = realloc(*ptrbuf, ((*i) + 1) * sizeof(struct
pixel));
324         ((*ptrbuf)[*i]).x = x1;
325         ((*ptrbuf)[(*i)++]).y = y1;
326         line_rect(image, x1, y1, rgba2);
327     }
328 }
329
330 void f_hex(struct PNG *image, int x, int y, union RGBA *rgba1, union
RGBA *rgba2){
331     struct pixel *ptrbuf = malloc(sizeof(struct pixel));
332     png_byte *ptr2;
333     int i = 0, x1, y1, n = 0;
334     ptrbuf[i].x = x;
335     ptrbuf[i++].y = y;
336     do {
337         x = ptrbuf[n].x;
338         y = ptrbuf[n++].y;
339         if (y > 0){
340             x1 = x;
341             y1 = y - 1;
342             fill_checker(image, &ptrbuf, &i, ptr2, x1, y1, rgba1,
rgba2);
343         }
344         if (y < image->height - 1){
345             x1 = x;
346             y1 = y + 1;
347             fill_checker(image, &ptrbuf, &i, ptr2, x1, y1, rgba1,
rgba2);
348         }
349         if (x > 0){
350             x1 = x - 1;
351             y1 = y;
```

```
352             fill_checker(image, &ptrbuf, &i, ptr2, x1, y1, rgba1,
rgba2);
353         }
354         if (x < image->width - 1){
355             x1 = x + 1;
356             y1 = y;
357             fill_checker(image, &ptrbuf, &i, ptr2, x1, y1, rgba1,
rgba2);
358         }
359     } while (n < i);
360     free(ptrbuf);
361 }
362
363 void copy(struct PNG *image, int x1, int y1, int x2, int y2, int x3,
int y3){
364     png_byte buf[y2-y1][(x2-x1)*4];
365     for (int i = 0; i < y2-y1; i++)
366         for (int j = 0; j < x2-x1; j++){
367             png_byte *ptr1 = &(image->row_pointers[i+y1][(j+x1) *
4]);
368             buf[i][j*4] = ptr1[0];
369             buf[i][j*4+1] = ptr1[1];
370             buf[i][j*4+2] = ptr1[2];
371             buf[i][j*4+3] = ptr1[3];
372         }
373     for (int i = 0; i < y2-y1; i++)
374         for (int j = 0; j < x2-x1; j++){
375             png_byte *ptr1 = &(image->row_pointers[i + y3][(j + x3)
* 4]);
376             png_byte *ptr2 = &buf[i][j*4];
377             ptr1[0] = ptr2[0];
378             ptr1[1] = ptr2[1];
379             ptr1[2] = ptr2[2];
380             ptr1[3] = ptr2[3];
381         }
382 }
383
384 int char_to_int(char *a) {
385     if (!a) {
386         return -1;
387     }
388     for (int i = 0; i < strlen(a); i++) {
389         if (!isdigit(a[i])) {
390             return -1;
391         }
392     }
393     return atoi(a);
394 }
```

## Chumak_Mikhail_cw/src/funcs.h

```
 1 #pragma once
 2
 3 #include "libs.h"
 4
 5 void help();
 6
 7 void init_png(struct PNG *image);
 8
 9 void img_info(struct PNG *image);
10
11 int read_png_file(char *file_name, struct PNG *image);
12
13 int write_png_file(char *file_name, struct PNG *image);
14
```

```
15 void clean_row(struct PNG* image);
16
17 void pixel(struct PNG *image, int x, int y, int thickness, union RGBA
* rgba1);
18
19 void line_hex(struct PNG *image, int x1, int y1, int x2, int y2, int
thickness, union RGBA * rgba1);
20
21 void line_rect(struct PNG *image, int x, int y, union RGBA *rgba1);
22
23 void rect(struct PNG *image, int x1, int y1, int x2, int y2, int
thickness, union RGBA *rgba1);
24
25 void fill_rect(struct PNG *image, int x1, int y1, int x2, int y2, int
thickness, union RGBA *rgba1, union RGBA *rgba2);
26
27 void hex(struct PNG *image, int x1, int y1, int x2, int y2, int
thickness, union RGBA *rgba1);
28
29 void fill_hex(struct PNG *image, int x1, int y1, int x2, int y2, int
thickness, union RGBA *rgba1, union RGBA *rgba2);
30
31 int cmp(const png_byte *ptr1, union RGBA *rgba1);
32
33 void fill_checker(struct PNG *image, struct pixel **ptrbuf, int *i,
png_byte *ptr2, int x1, int y1, union RGBA *rgba1, union RGBA *rgba2);
34
35 void f_hex(struct PNG *image, int x, int y, union RGBA *rgba1, union
RGBA *rgba2);
36
37 void copy(struct PNG *image, int x1, int y1, int x2, int y2, int x3,
int y3);
38
39 int char_to_int(char *a);
```

## Chumak_Mikhail_cw/src/libs.h

```
 1 #pragma once
 2
 3 #include <stdio.h>
 4 #include <stdlib.h>
 5 #include <getopt.h>
 6 #include <ctype.h>
 7 #include <string.h>
 8 #include <locale.h>
 9 #include <png.h>
10 #include <math.h>
11
12 union RGBA{
13     png_byte ptr[4];
14 };
15
16 struct pixel{
17     int x;
18     int y;
19 };
20
21 struct PNG{
22     int width;
23     int height;
24     png_byte color_type;
25     png_byte bit_depth;
26     png_structp png_ptr;
27     png_infop info_ptr;
28     int number_of_passes;
```

```
29     png_bytep *row_pointers;
30 };
```

**Комментарии из пулл-реквестов**

Исходный код:

```
    image.row_pointers = NULL;
    image.number_of_passes = 0;
    image.info_ptr = NULL;
    image.color_type = 0;
    image.bit_depth = 0;
    image.png_ptr = NULL;
    image.width = 0;
    image.height = 0;
    struct PNG image;

    char *input = NULL, *output = NULL, *state = NULL;
    rgba2.ptr[0] = 0, rgba2.ptr[1] = 0, rgba2.ptr[2] = 0, rgba2.ptr[3] = 0;
    rgba1.ptr[0] = 0, rgba1.ptr[1] = 0, rgba1.ptr[2] = 0, rgba1.ptr[3] = 0;
```

Комментарии:

NataRazmochaeva:Я бы оформила в виде какой-нибудь init_png() функции. На Ваше усмотрение.

Machumak00:Да, так удобнее. Сделал

Исходный код:

```
        int d = -lengthX;
        int y = y1;
        int x = x1;
    if (lengthY <= lengthX){
        pixel(image, x1, y1, thickness, rgba1);
    if (length == 0)
        length = lengthY;
    else
        length = lengthX;
    if (lengthX > lengthY)
    int length;
    int lengthY = abs(y2 - y1);
    int lengthX = abs(x2 - x1);
    int dy = (y2 - y1 >= 0 ? 1 : -1);
    int dx = (x2 - x1 >= 0 ? 1 : -1);
void line_hex(struct PNG *image, int x1, int y1, int x2, int y2, int thickness,
union RGBA * rgba1){

}
        }
           }
              ptr[3] = rgba1->ptr[3];
              ptr[2] = rgba1->ptr[2];
              ptr[1] = rgba1->ptr[1];
              ptr[0] = rgba1->ptr[0];
              png_byte *ptr = &(image->row_pointers[y+i][(x+j)*4]);
           if(y+i >= 0 && y+i < image->height && x+j >= 0 && x+j < image-
>width && i*i+j*j<=thickness*thickness/4){
        for(int j = -thickness/2; j < thickness - thickness/2; j++){
    for(int i = -thickness/2; i < thickness - thickness/2; i++)
void pixel(struct PNG *image, int x, int y, int thickness, union RGBA *
rgba1){

}
    free(image->row_pointers);
        free(image->row_pointers[i]);
    for (int i = 0; i < image->height; i++)
void clean_row(struct PNG* image){

}
    return 0;

    fclose(fp);

    free(image->row_pointers);
        free(image->row_pointers[y]);
```

```
    for (y = 0; y < image->height; y++)

    png_write_end(image->png_ptr, NULL);

    }
       return 1;
       fclose(fp);
       printf("Error writing file.");
       png_destroy_write_struct(&image->png_ptr, &image->info_ptr);
    if (setjmp(png_jmpbuf(image->png_ptr))){

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

Комментарии:
NataRazmochaeva:Можно вынести объявление переменных в начало функции?
Machumak00:Вынес
Исходный код:

```
        }
            }
                x += dx;
                d -= 2 * lengthY;
            if (d > 0){
            d += 2 * lengthX;
            y += dy;
            pixel(image, x, y, thickness, rgba1);
        while(length--){
        length++;
        int d = - lengthY;
        int y = y1;
        int x = x1;
    } else{
        }
            }
                y += dy;
                d -= 2 * lengthX;
            if (d > 0){
            d += 2 * lengthY;
            x += dx;
            pixel(image, x, y, thickness, rgba1);
        while(length--){
        length++;
        int d = -lengthX;
        int y = y1;
        int x = x1;
    if (lengthY <= lengthX){
        pixel(image, x1, y1, thickness, rgba1);
    if (length == 0)
        length = lengthY;
    else
        length = lengthX;
    if (lengthX > lengthY)
    int length;
    int lengthY = abs(y2 - y1);
    int lengthX = abs(x2 - x1);
    int dy = (y2 - y1 >= 0 ? 1 : -1);
    int dx = (x2 - x1 >= 0 ? 1 : -1);
void line_hex(struct PNG *image, int x1, int y1, int x2, int y2, int thickness,
union RGBA * rgba1){

}
        }
            }
                ptr[3] = rgba1->ptr[3];
```

```
            ptr[2] = rgba1->ptr[2];
            ptr[1] = rgba1->ptr[1];
            ptr[0] = rgba1->ptr[0];
            png_byte *ptr = &(image->row_pointers[y+i][(x+j)*4]);
        if(y+i >= 0 && y+i < image->height && x+j >= 0 && x+j < image-
>width && i*i+j*j<=thickness*thickness/4){
      for(int j = -thickness/2; j < thickness - thickness/2; j++){
   for(int i = -thickness/2; i < thickness - thickness/2; i++)
void pixel(struct PNG *image, int x, int y, int thickness, union RGBA *
rgba1){

}
   free(image->row_pointers);
      free(image->row_pointers[i]);
   for (int i = 0; i < image->height; i++)
void clean_row(struct PNG* image){

}
   return 0;
```

Комментарии:

NataRazmochaeva:Код похож с тем, что в ветке true. Можно оформить его в виде функции?

Machumak00:Оформил в виде функции line_vd_hex()

Исходный код:

```
    line_hex(image, a6[0], a6[1], a1[0], a1[1], thickness, rgba1);
    line_hex(image, a5[0], a5[1], a6[0], a6[1], thickness, rgba1);
    line_hex(image, a4[0], a4[1], a5[0], a5[1], thickness, rgba1);
    line_hex(image, a3[0], a3[1], a4[0], a4[1], thickness, rgba1);
    line_hex(image, a2[0], a2[1], a3[0], a3[1], thickness, rgba1);
    line_hex(image, a1[0], a1[1], a2[0], a2[1], thickness, rgba1);
    a6[0] = (int)(center[0]+radius*cos(M_PI/6)); a6[1] = center[1]-radius/2;
    a5[0] = (int)(center[0]+radius*cos(M_PI/6)); a5[1] = center[1]+radius/2;
    a4[0] = center[0]; a4[1] = center[1]+radius;
    a3[0] = (int)(center[0]-radius*cos(M_PI/6)); a3[1] = center[1]+radius/2;
    a2[0] = (int)(center[0]-radius*cos(M_PI/6)); a2[1] = center[1]-radius/2;
    a1[0] = center[0]; a1[1] = center[1]-radius;
    int a1[2], a2[2], a3[2], a4[2], a5[2], a6[2];
    int radius = (x2-x1)/2;
    int center[2] = {(x2+x1)/2, (y2+y1)/2};
void hex(struct PNG *image, int x1, int y1, int x2, int y2, int thickness, union
RGBA *rgba1){

}
    }
        }
            line_rect(image, x, y, rgba2);
        for (int x = x1 + thickness; x < x2 - thickness + 1; x++) {
    for (int y = y1 + thickness; y < y2 - thickness + 1; y++) {
        }
            }
                line_rect(image, x, y, rgba1);
            if (x < x1 + thickness || x > x2 - thickness || y < y1 + thickness || y >
y2 - thickness){
        for (int x = x1; x <= x2; x++){
    for (int y = y1; y <= y2; y++)
void fill_rect(struct PNG *image, int x1, int y1, int x2, int y2, int thickness,
union RGBA *rgba1, union RGBA *rgba2){

}
        }
            }
                line_rect(image, x, y, rgba1);
            if (x < x1 + thickness || x > x2 - thickness || y < y1 + thickness || y >
y2 - thickness){
        for (int x = x1; x <= x2; x++){
    for (int y = y1; y <= y2; y++)
void rect(struct PNG *image, int x1, int y1, int x2, int y2, int thickness, union
RGBA *rgba1) {

}
```

```
    ptr[3] = rgba1->ptr[3];
    ptr[2] = rgba1->ptr[2];
    ptr[1] = rgba1->ptr[1];
    ptr[0] = rgba1->ptr[0];
    png_byte *ptr = &(image->row_pointers[y][x * 4]);
void line_rect(struct PNG *image, int x, int y, union RGBA *rgba1){

}
    }
        }
            }
                x += dx;
                d -= 2 * lengthY;
            if (d > 0){
            d += 2 * lengthX;
            y += dy;
            pixel(image, x, y, thickness, rgba1);
        while(length--){
        length++;
        int d = - lengthY;
        int y = y1;
        int x = x1;
    } else{
        }
            }
                y += dy;
                d -= 2 * lengthX;
            if (d > 0){
            d += 2 * lengthY;
            x += dx;
            pixel(image, x, y, thickness, rgba1);
        while(length--){
        length++;
        int d = -lengthX;
```

Комментарии:

NataRazmochaeva:Возможно ли здесь использовать двумерную матрицу (чтобы, например, строки 295-300 оформить в виде цикла ) ?

Machumak00:Создал двумерный массив arr[6][2] и реализовал функцию в цикле

Исходный код:

```
            line_rect(image, x1, y1, rgba2);
            ptrbuf[i++].y = y1;
            ptrbuf[i].x = x1;
            ptrbuf = realloc(ptrbuf, (i + 1) * sizeof(struct pixel));
         if (cmp(ptr2, rgba1) && cmp(ptr2, rgba2)) {
         ptr2 = &(image->row_pointers[y1][x1 * 4]);
         y1 = y + 1;
         x1 = x;
      if (y < image->height - 1) {
      }
         }
            line_rect(image, x1, y1, rgba2);
            ptrbuf[i++].y = y1;
            ptrbuf[i].x = x1;
            ptrbuf = realloc(ptrbuf, (i + 1) * sizeof(struct pixel));
         if (cmp(ptr2, rgba1) && cmp(ptr2, rgba2)) {
         ptr2 = &(image->row_pointers[y1][x1 * 4]);
         y1 = y - 1;
         x1 = x;
      if (y > 0) {
      y = ptrbuf[n++].y;
      x = ptrbuf[n].x;
   do {
   ptrbuf[i++].y = y;
   ptrbuf[i].x = x;
   int i = 0, x1, y1, n = 0;
   png_byte *ptr2;
   struct pixel *ptrbuf = calloc(1, sizeof(struct pixel));
void f_hex(struct PNG *image, int x, int y, union RGBA *rgba1, union
RGBA *rgba2) {

}
   else return 1;
      return 0;
      ptr1[3] == rgba1->ptr[3])
   if (ptr1[0] == rgba1->ptr[0] && ptr1[1] == rgba1->ptr[1] && ptr1[2] ==
rgba1->ptr[2] &&
int cmp(const png_byte *ptr1, union RGBA *rgba1){

}
   f_hex(image, x2 - (x2 - x1) / 2, y2 - (y2 - y1) / 2, rgba1, rgba2);
   hex(image, x1, y1, x2, y2, thickness, rgba1);
void fill_hex(struct PNG *image, int x1, int y1, int x2, int y2, int thickness,
union RGBA *rgba1, union RGBA *rgba2){

}
```

```
    line_hex(image, a6[0], a6[1], a1[0], a1[1], thickness, rgba1);
    line_hex(image, a5[0], a5[1], a6[0], a6[1], thickness, rgba1);
    line_hex(image, a4[0], a4[1], a5[0], a5[1], thickness, rgba1);
    line_hex(image, a3[0], a3[1], a4[0], a4[1], thickness, rgba1);
    line_hex(image, a2[0], a2[1], a3[0], a3[1], thickness, rgba1);
    line_hex(image, a1[0], a1[1], a2[0], a2[1], thickness, rgba1);
    a6[0] = (int)(center[0]+radius*cos(M_PI/6)); a6[1] = center[1]-radius/2;
    a5[0] = (int)(center[0]+radius*cos(M_PI/6)); a5[1] = center[1]+radius/2;
    a4[0] = center[0]; a4[1] = center[1]+radius;
    a3[0] = (int)(center[0]-radius*cos(M_PI/6)); a3[1] = center[1]+radius/2;
    a2[0] = (int)(center[0]-radius*cos(M_PI/6)); a2[1] = center[1]-radius/2;
    a1[0] = center[0]; a1[1] = center[1]-radius;
    int a1[2], a2[2], a3[2], a4[2], a5[2], a6[2];
    int radius = (x2-x1)/2;
    int center[2] = {(x2+x1)/2, (y2+y1)/2};
void hex(struct PNG *image, int x1, int y1, int x2, int y2, int thickness, union
RGBA *rgba1){

}
    }
        }
            line_rect(image, x, y, rgba2);
        for (int x = x1 + thickness; x < x2 - thickness + 1; x++) {
    for (int y = y1 + thickness; y < y2 - thickness + 1; y++) {
        }
            }
                line_rect(image, x, y, rgba1);
            if (x < x1 + thickness || x > x2 - thickness || y < y1 + thickness || y >
y2 - thickness){
        for (int x = x1; x <= x2; x++){
    for (int y = y1; y <= y2; y++)
void fill_rect(struct PNG *image, int x1, int y1, int x2, int y2, int thickness,
union RGBA *rgba1, union RGBA *rgba2){

}
        }
            }
                line_rect(image, x, y, rgba1);
            if (x < x1 + thickness || x > x2 - thickness || y < y1 + thickness || y >
y2 - thickness){
        for (int x = x1; x <= x2; x++){
    for (int y = y1; y <= y2; y++)
void rect(struct PNG *image, int x1, int y1, int x2, int y2, int thickness, union
RGBA *rgba1) {

}
```

```
    ptr[3] = rgba1->ptr[3];
    ptr[2] = rgba1->ptr[2];
```
Комментарии:
NataRazmochaeva:Повторяющийся сценарий работы. Можно сделать в виде отдельной функции?
Machumak00:Оформил в виде функции fill_checker()

Изменения:
```
+void init_png(struct PNG *image){
+    image->height = 0;
+    image->width = 0;
+    image->png_ptr = NULL;
+    image->bit_depth = 0;
+    image->color_type = 0;
+    image->info_ptr = NULL;
+    image->number_of_passes = 0;
+    image->row_pointers = NULL;
+}
+

-int write_png_file(char *file_name, struct PNG *image) {
+int write_png_file(char *file_name, struct PNG *image){

+void line_vd_hex(struct PNG *image, int x, int y, int length, int lengthX, int lengthY, int dx, int dy, int thickness, union RGBA * rgba1, int k){
+    int d = -lengthX;
+    length++;
+    while(length--){
+        if (k == 0)
+            pixel(image, x, y, thickness, rgba1);
+        else
+            pixel(image, y, x, thickness, rgba1);
+        x += dx;
+        d += 2 * lengthY;
+        if (d > 0){
+            d -= 2 * lengthX;
+            y += dy;
+        }
+    }
+}
+

+    int x = x1, y = y1, k = 0;
```

```
-    if (lengthY <= lengthX){
-        int x = x1;
-        int y = y1;
-        int d = -lengthX;
-        length++;
-        while(length--){
-            pixel(image, x, y, thickness, rgba1);
-            x += dx;
-            d += 2 * lengthY;
-            if (d > 0){
-                d -= 2 * lengthX;
-                y += dy;
-            }
-        }
-    } else{
-        int x = x1;
-        int y = y1;
-        int d = - lengthY;
-        length++;
-        while(length--){
-            pixel(image, x, y, thickness, rgba1);
-            y += dy;
-            d += 2 * lengthX;
-            if (d > 0){
-                d -= 2 * lengthY;
-                x += dx;
-            }
-        }
+    if (lengthY <= lengthX)
+        line_vd_hex(image, x, y, length, lengthX, lengthY, dx, dy, thickness,
rgba1, k);
+    else {
+        k = 1;
+        line_vd_hex(image, y, x, length, lengthY, lengthX, dy, dx, thickness,
rgba1, k);
```