

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

КУРСОВАЯ РАБОТА
по дисциплине «Программирование»
Тема: Работа с PNG файлами.

Студентка гр. 9381

Соболева К.С.

Преподаватель

Берленко Т.А.

Санкт-Петербург

2020

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент: Соболева К.С.

Группа: 9381

Тема работы: Работа с PNG файлами.

Исходные данные: Язык программирования C. Библиотека libpng. Компилятор GCC.

Содержание пояснительной записки:

«Задание»

«Введение»

«Содержание»

«Заключение»

«Список использованных источников»

Предполагаемый объем пояснительной записки:

Не менее 20 страниц.

Дата выдачи задания: 01.03.2020

Дата сдачи реферата: 27.05.2020

Дата защиты реферата: 27.05.2020

Студентка

Соболева К.С.

Преподаватель

Берленко Т.А.

АННОТАЦИЯ

Разработана программа для обработки изображения в формате PNG согласно действиям пользователя. Реализован CLI, парсинг команд выполнен при помощи функции `getopt()`. Доступна справка о программе, информация об изображении и его обработка.

SUMMARY

A program has been developed for processing PNG images according to user actions. The CLI is implemented, command parsing is performed using the `getopt()` function. Information about the image and its processing are available.

	Введение	5
1.	Исходное задание	6
2.	Функции считывания и сохранения изображения	7
2.1	Функция main	7
2.2	Функция readPng	7
2.3	Функция writePng	7
3.	Функции обработки изображения	8
3.1	Функция changeColor	8
3.2	Функция создания рамки frameLine	8
3.3	Функция создания рамки gradient	8
3.4	Функция обводки прямоугольников rectangles	8
4.	Тестирование программы.	10
	Заключение	12
	Список использованных источников	12
	Приложение А. Исходный код	12

ВВЕДЕНИЕ

Цель работы.

Обработка изображения в формате PNG согласно действиям пользователя.

Задачи.

- Изучение синтаксиса языка программирования C.
- Изучение библиотеки libpng.
- Написание исходного кода.
- Проверка работоспособности программы.
- Редактирование кода.
- Итоговое тестирование программы.

1. ИСХОДНОЕ ЗАДАНИЕ

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

- (1) Заменяет все пиксели одного заданного цвета на другой цвет.
Функционал определяется:
 - Цвет, который требуется заменить
 - Цвет на который требуется заменить
- (2) Сделать рамку в виде узора. Рамка определяется:
 - Узором (должно быть несколько на выбор. Красивый узор можно получить используя фракталы)
 - Цветом
 - Шириной
- (3) Поиск всех залитых прямоугольников заданного цвета.
Требуется найти все прямоугольники заданного цвета и обвести их линиями. Функционал определяется:
 - Цветом искомых прямоугольников
 - Цветом линии для обводки
 - Толщиной линии для обводки

2. Функции считывания и сохранения изображения

2.1 Функция main

Данная функция обрабатывает аргументы, передаваемые программе на вход. Для парсинга параметров используется функция `getopt_long` и оператор `switch`. Объявлена строка `shortOpts` для коротких опций. Для обнуления полей структуры `Opts` используется функция `memset()`.

2.2 Функция readPng

Функция для считывания изображения из файла, название которого передаётся в качестве аргумента. Используются функции библиотеки `libpng`, обработка ошибок реализована при помощи функции `setjmp()`. В случае некорректного изображения программа выведет соответствующее сообщение и прекратит свою работу. Все данные считываются в структуру `Png`, поля которой хранят все необходимые данные об изображении.

2.3 Функция writePng

Функция для записи изображения в файл. Аналогично считыванию, используются функции библиотеки `libpng`, а обработка ошибок реализована при помощи функции `setjmp()`.

3. Функции обработки изображения

3.1 Функция `changeColor`

Данная функция предназначена для замены всех пикселей одного цвета на другой. Происходит простой обход всего изображения и проверка каждого пикселя на соответствие введённому цвету. Если цвет совпадает, то пиксель заменяется. Используются вспомогательные функции `isMatch()` и `setPixel()` для сравнения цвета и замены пикселя соответственно.

3.2. Функция создания рамки `frameLine`

Данная функция рисует рамку вокруг изображения в виде градиента, с пропуском некоторых позиций. Используется тот же алгоритм, что и в функции `gradient()`.

3.3. Функция создания рамки `gradient`

Данная функция рисует рамку вокруг изображения. Рамка представляет собой градиент, цвет которого выбирает сам пользователь. Цвет “раскрывается” сверху-вниз, реализовано при помощи расчёта текущего положения и выбора подходящего оттенка соответственно “высоте”.

3.4. Функция обводки прямоугольников `rectangles`

Данная функция обводит все прямоугольники заданного цвета линией с заданной толщиной и цветом. Реализован следующий алгоритм:

1. Производим обход всего изображения, ищем прямоугольники (сравниваем текущий цвет с необходимым) и формируем “карту”, записанную в массив `int` (прямоугольники в нём представлены единицами, а всё остальное нулями).
2. Обходим нашу “карту”. Для обводки прямоугольника нам необходимо знать его верхнюю левую и нижнюю правую точки. Для их поиска мы проходим весь массив, и, если наткнемся на единицу (а это обязана быть левая верхняя точка прямоугольника), то идём сначала до верхней правой, а затем и до

нижней правой точки. По пути заменяем все единицы двойками, чтобы не работать с тем же самым прямоугольником несколько раз. Запоминаем необходимые нам точки.

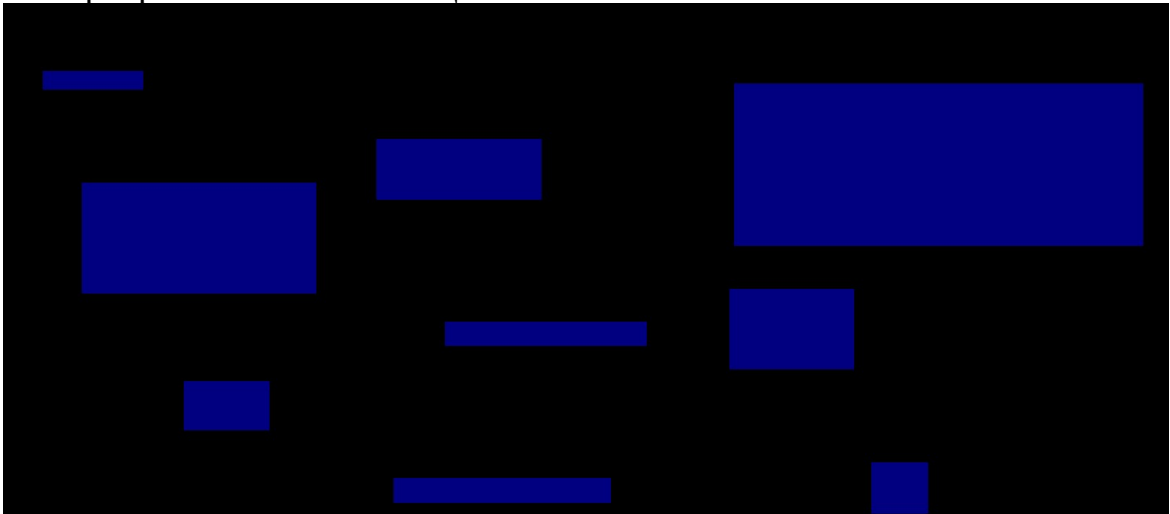
3. Обводим прямоугольники линиями.

4. Тестирование программы

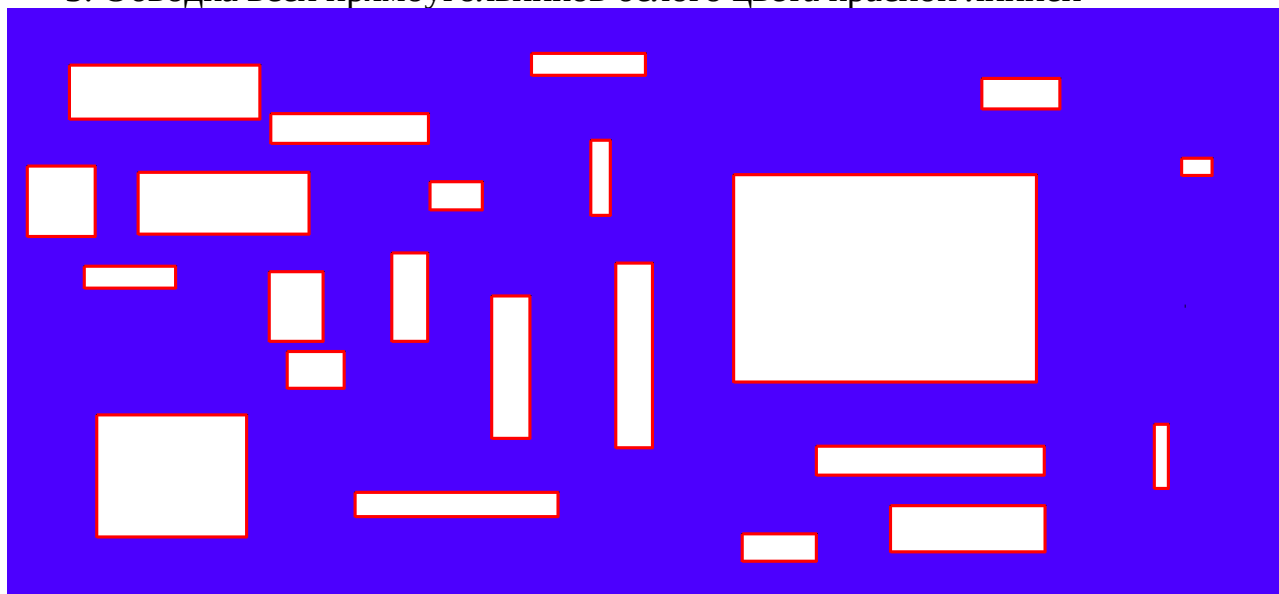
1. Рисование рамки второго типа красного цвета



2. Перекрашивание белого цвета в тёмно-синий



3. Обводка всех прямоугольников белого цвета красной линией



ЗАКЛЮЧЕНИЕ

В ходе выполнения данной курсовой работы был изучен синтаксис языка программирования C, библиотека `libpng`.

Реализован следующий функционал:

1. Замена всех пикселей одного цвета пикселем другого цвета.
2. Обводка всех прямоугольников заданного цвета линией заданного цвета и толщины.
3. Рисование рамки вокруг изображения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Керниган Б. И Ритчи д. Язык программирования Си М.: Вильямс, 1978, 288 с.

Приложение А

Исходный код

/source/main.c

```
1 #include <stdio.h>
2 #include <getopt.h>
3 #include <string.h>
4 #include "../include/png_image.h"
5 #include "../include/structures.h"
6
7 static struct option longOpts[] = {
8     {"info",      no_argument,      NULL, 'i'},
9     {"help",      no_argument,      NULL, 'h'},
10    {"rect",       no_argument,      NULL, 'r'},
11    {"color",      no_argument,      NULL, 'c'},
12    {"paint",      no_argument,      NULL, 'p'},
13    {"width",      required_argument, NULL, 's'},
14    {"write",      required_argument, NULL, 'w'},
15    {"frame",      required_argument, NULL, 'F'},
16    {"from",       required_argument, NULL, 'f'},
17    {"to",         required_argument, NULL, 't'},
18    {NULL,        0,                 NULL, 0}
19 };
20
21 void printHelp() {
22     printf("Справка по использованию программы: \n"
23         "По умолчанию изображение будет сохранено в тот же файл,
из которого оно было считано."
24         " Для выбора другого файла, пожалуйста, воспользуйтесь
флагом -w / --write.\n"
25         "Информация об изображении:                -i /
--info\n"
26         "Данная справка:                            -h /
--help\n"
27         "Выбора цвета:                                -c /
--color\n"
28         "Цвет, который требуется поменять:              -f /
--from\n"
29         "Цвет, на который требуется поменять:            -t /
--to\n"
30         "Поиск всех прямоугольников заданного цвета:    -r /
--rect\n"
31         "Замена всех пикселей одного цвета на другой цвет: -p /
--paint\n"
32         "Рамка вокруг изображения:                        -F /
--frame\n"
33         "Толщина рамки:                                    -s /
--width\n"
34         "Толщина прямой для обводки прямоугольников(1-9): -T\n"
35         "Цвет рамки:                                       -C\n"
36         "Цвет для обводки прямоугольников выбирается флагом -c,
например: -r --color -f white -t navy\n"
37         "Доступные цвета: \t\t Доступные рамки: \n"
38         "red                                     1 - градиент\n"
39         "green                                    2 - выколотые точки\n"
40         "blue\n"
41         "navy\n"
42         "aqua\n"
43         "white\n"
44         "ivory\n");
45 }
46
```

```

47 void cleanMemory(Png *image, Opts *options) {
48     for (int y = 0; y < image->height; y++)
49         free(image->row_pointers[y]);
50     free(image->row_pointers);
51     png_destroy_read_struct(&image->png_ptr, NULL, NULL);
52     free(image->info_ptr);
53     free(image);
54     if (options->colorFrom)
55         free(options->colorFrom);
56     if (options->colorTo)
57         free(options->colorTo);
58     if (options->frameColor)
59         free(options->frameColor);
60 }
61
62 png_bytep initColor(const char *name) {
63     png_bytep color = calloc(3, sizeof(unsigned int));
64     if (!name) {
65         free(color);
66         return NULL;
67     }
68     if (!strcmp(name, "red")) {
69         color[0] = 255;
70         return color;
71     }
72     if (!strcmp(name, "green")) {
73         color[1] = 255;
74         return color;
75     }
76     if (!strcmp(name, "blue")) {
77         color[2] = 255;
78         return color;
79     }
80     if (!strcmp(name, "navy")) {
81         color[2] = 128;
82         return color;
83     }
84     if (!strcmp(name, "white")) {
85         color[0] = 255;
86         color[1] = 255;
87         color[2] = 255;
88         return color;
89     }
90     if (!strcmp(name, "aqua")) {
91         color[1] = 255;
92         color[2] = 255;
93         return color;
94     }
95     if (!strcmp(name, "ivory")) {
96         color[0] = 255;
97         color[1] = 255;
98         color[2] = 240;
99         return color;
100    }
101    free(color);
102    return NULL;
103 }
104
105 int main(int argc, char *argv[]) {
106     if (argc == 1) {
107         printHelp();
108         return 0;
109     }
110

```

```

111     const char *shortOpts = "irchF:s:p:w:f:t:d:C:T:";
112     int opt = -1;
113     int longIndex = 0;
114
115     Opts options;
116     memset(&options, 0, sizeof(Opts));
117     options.thick = 1;
118
119     while ((opt = getopt_long(argc, argv, shortOpts, longOpts,
120 &longIndex)) != -1) {
121         switch(opt) {
122             case 'T':
123                 if (sscanf(optarg, "%d", &options.thick) != 1) {
124                     printHelp();
125                     return 1;
126                 }
127                 break;
128             case 'r':
129                 if (!argv[optind]) {
130                     printHelp();
131                     return 1;
132                 }
133                 if (!strcmp(argv[optind], "--color") && !
134 strcmp(argv[optind], "-c")) {
135                     printHelp();
136                     return 1;
137                 }
138                 options.rect = 1;
139                 break;
140             case 'F':
141                 if (sscanf(optarg, "%d", &options.frameType) != 1) {
142                     printHelp();
143                     return 1;
144                 }
145                 if (options.frameType != GRADIENT &&
146 options.frameType != LINES) {
147                     printHelp();
148                     return 1;
149                 }
150                 options.frame = 1;
151                 break;
152             case 'w':
153                 if (!optarg) {
154                     printHelp();
155                     return 1;
156                 }
157                 options.fileWrite = optarg;
158                 break;
159             case 'p':
160                 if (!argv[optind]) {
161                     printHelp();
162                     return 1;
163                 }
164                 if (!strcmp(argv[optind], "--color") && !
165 strcmp(argv[optind], "-c")) {
166                     printHelp();
167                     return 1;
168                 }
169                 options.color = 1;
170                 options.repaint = 1;
171                 break;
172             case 's':
173                 if (sscanf(optarg, "%d", &options.width) != 1) {
174                     printHelp();
175                     return 1;
176                 }
177                 break;
178             default:
179                 printHelp();
180                 return 1;
181         }
182     }

```



```

171         return 1;
172     }
173     break;
174 case 'i':
175     options.info = 1;
176     break;
177 case 'C':
178     if (!options.frame) {
179         printHelp();
180         return 1;
181     }
182     options.frameColor = initColor(optarg);
183     if (!options.frameColor) {
184         printHelp();
185         return 1;
186     }
187     break;
188 case 'c':
189     if (options.repaint != 1 && options.rect != 1) {
190         printHelp();
191         return 1;
192     }
193     options.color = 1;
194     break;
195 case 'f':
196     if (options.color != 1 || optarg == NULL) {
197         printHelp();
198         return 1;
199     }
200     options.colorFrom = initColor(optarg);
201     if (!options.colorFrom) {
202         printHelp();
203         return 1;
204     }
205     options.to = 1;
206     break;
207 case 't':
208     if (options.to != 1 || optarg == NULL) {
209         printHelp();
210         return 1;
211     }
212     options.colorTo = initColor(optarg);
213     if (!options.colorTo) {
214         printHelp();
215         free(options.colorFrom);
216         return 1;
217     }
218     break;
219 case '?':
220 case 'h':
221     default:
222         printHelp();
223         return 1;
224     }
225 }
226
227 const char *savePath = argv[argc-1];
228 Png *image = readPng(savePath);
229 if (!image) {
230     printf("Изображение не удалось считать.\n");
231     printHelp();
232     return 1;
233 }
234

```

```

235     if (options.repaint)
236         changeColor(options.colorTo, options.colorFrom, image);
237     if (options.info)
238         printInfo(image);
239     if (options.frame) {
240         switch (options.frameType) {
241             case GRADIENT:
242                 gradient(options.frameColor, options.width, image);
243                 break;
244             case LINES:
245                 frameLine(options.frameColor, options.width, image);
246                 break;
247             default:
248                 printHelp();
249                 break;
250         }
251     }
252     if (options.rect && options.colorTo && options.colorFrom)
253         rectangles(options.thick, options.colorTo,
options.colorFrom, image);
254     if (options.fileWrite)
255         savePath = options.fileWrite;
256
257     writePng(savePath, image);
258     cleanMemory(image, &options);
259     return 0;
260 }

```

/source/png_image.c

```

1  #include "../include/png_image.h"
2
3  Png* readPng(const char *filePath) {
4      FILE *pFile = fopen(filePath, "rb");
5      if (!pFile) {
6          printf("Не удалось открыть файл для считывания.\n");
7          return NULL;
8      }
9
10     png_byte header[8];
11     fread(header, sizeof(png_byte), 8, pFile);
12     if (png_sig_cmp(header, 0, 8)) {
13         fclose(pFile);
14         return NULL;
15     }
16
17     Png *image = malloc(sizeof(Png));
18     image->png_ptr = png_create_read_struct(PNG_LIBPNG_VER_STRING,
NULL, NULL, NULL);
19
20     if (!image->png_ptr) {
21         free(image);
22         fclose(pFile);
23         return NULL;
24     }
25
26     image->info_ptr = png_create_info_struct(image->png_ptr);
27     if (!image->info_ptr) {
28         png_destroy_read_struct(&image->png_ptr, NULL, NULL);
29         fclose(pFile);
30         free(image);
31         return NULL;
32     }
33
34     if (setjmp(png_jmpbuf(image->png_ptr))) {

```

```

35         png_destroy_read_struct(&image->png_ptr, NULL, NULL);
36         fclose(pFile);
37         free(image);
38         return NULL;
39     }
40
41     png_init_io(image->png_ptr, pFile);
42     png_set_sig_bytes(image->png_ptr, 8);
43
44     png_read_info(image->png_ptr, image->info_ptr);
45
46     image->width = png_get_image_width(image->png_ptr, image-
47 >info_ptr);
48     image->height = png_get_image_height(image->png_ptr, image-
49 >info_ptr);
50     image->color_type = png_get_color_type(image->png_ptr, image-
51 >info_ptr);
52     image->bit_depth = png_get_bit_depth(image->png_ptr, image-
53 >info_ptr);
54     image->number_of_passes = png_set_interlace_handling(image-
55 >png_ptr);
56     png_read_update_info(image->png_ptr, image->info_ptr);
57
58     if (setjmp(png_jmpbuf(image->png_ptr))) {
59         png_destroy_read_struct(&image->png_ptr, NULL, NULL);
60         fclose(pFile);
61         free(image);
62         return NULL;
63     }
64
65     if (image->bit_depth != 8 || (image->color_type !=
66 PNG_COLOR_TYPE_RGB &&
67 image->color_type != PNG_COLOR_TYPE_RGBA)) {
68     printf("Данное изображение не поддерживается.\n");
69     png_destroy_read_struct(&image->png_ptr, NULL, NULL);
70     fclose(pFile);
71     return NULL;
72 }
73
74     image->row_pointers = malloc(sizeof(png_bytep) * image->height);
75     for (int y = 0; y < image->height; y++)
76         image->row_pointers[y] = malloc(png_get_rowbytes(image-
77 >png_ptr, image->info_ptr));
78
79     png_read_image(image->png_ptr, image->row_pointers);
80
81     fclose(pFile);
82     return image;
83 }
84
85 void writePng(const char *filePath, Png *image) {
86     if (!image || !filePath)
87         return;
88
89     FILE *pFile = fopen(filePath, "wb");
90     if (!pFile)
91         return;
92
93     png_structp pngStructPtr =
94 png_create_write_struct(PNG_LIBPNG_VER_STRING,
95 NULL, NULL, NULL);
96     if (!pngStructPtr) {
97         fclose(pFile);

```

```

91         return;
92     }
93
94     png_infop pngInfoStructPtr =
png_create_info_struct(pngStructPtr);
95     if (!pngInfoStructPtr) {
96         png_destroy_write_struct(&pngStructPtr, NULL);
97         fclose(pFile);
98         return;
99     }
100
101     if (setjmp(png_jmpbuf(pngStructPtr))) {
102         png_destroy_write_struct(&pngStructPtr, &pngInfoStructPtr);
103         fclose(pFile);
104         return;
105     }
106
107     png_init_io(pngStructPtr, pFile);
108
109     if (setjmp(png_jmpbuf(pngStructPtr))) {
110         png_destroy_write_struct(&pngStructPtr, &pngInfoStructPtr);
111         fclose(pFile);
112         return;
113     }
114
115     png_set_IHDR(pngStructPtr, pngInfoStructPtr, image->width,
image->height, 8, PNG_COLOR_TYPE_RGBA,
116                 PNG_INTERLACE_NONE, PNG_COMPRESSION_TYPE_BASE,
PNG_FILTER_TYPE_BASE);
117     png_write_info(pngStructPtr, pngInfoStructPtr);
118
119     if (setjmp(png_jmpbuf(pngStructPtr))) {
120         png_destroy_write_struct(&pngStructPtr, &pngInfoStructPtr);
121         fclose(pFile);
122         return;
123     }
124
125     png_write_image(pngStructPtr, image->row_pointers);
126
127     if (setjmp(png_jmpbuf(pngStructPtr))) {
128         png_destroy_write_struct(&pngStructPtr, &pngInfoStructPtr);
129         fclose(pFile);
130         return;
131     }
132
133     png_write_end(pngStructPtr, NULL);
134     png_destroy_write_struct(&pngStructPtr, &pngInfoStructPtr);
135     fclose(pFile);
136 }
137
138 int isMatch(png_bytep colorA, png_bytep colorB) {
139     if (!colorA || !colorB)
140         return 0;
141     for (int i = 0; i < 3; i++)
142         if (colorA[i] != colorB[i])
143             return 0;
144     return 1;
145 }
146
147 void setPixel(png_bytep pixel, png_bytep color) {
148     for (int i = 0; i < 3; i++)
149         pixel[i] = color[i];
150 }
151

```

```

152 void changeColor(png_bytep colorTo, png_bytep colorFrom, Png *image)
153 {
154     for (int y = 0; y < image->height; y++)
155         for (int x = 0; x < image->width; x++) {
156             png_bytep px = &(image->row_pointers[y][x*4]);
157             if (isMatch(px, colorFrom))
158                 setPixel(px, colorTo);
159         }
160 }
161 void drawLine(int x1, int y1, int x2, int y2, png_bytep color, Png*
image) {
162     const int deltaX = abs(x2 - x1);
163     const int deltaY = abs(y2 - y1);
164     const int signX = x1 < x2 ? 1 : -1;
165     const int signY = y1 < y2 ? 1 : -1;
166     int error = deltaX - deltaY;
167
168     setPixel(&(image->row_pointers[y2][x2*4]), color);
169     while (x1 != x2 || y1 != y2) {
170         setPixel(&(image->row_pointers[y1][x1*4]), color);
171         const int error2 = error * 2;
172         if (error2 > -deltaY) {
173             error -= deltaY;
174             x1 += signX;
175         }
176         if (error2 < deltaX) {
177             error += deltaX;
178             y1 += signY;
179         }
180     }
181 }
182
183 void drawLineThick(int x1, int y1, int x2, int y2, int thick,
png_bytep color, Png* image) {
184     if (x1 < 0 || x2 < 0 || y1 < 0 || y2 < 0 || x1 > image->width ||
x2 > image->width || y2 > image->height ||
185         y1 > image->height || x1-thick/2 < 0 || x2-thick/2 < 0 ||
y1-thick/2 < 0 || y2-thick/2 < 0 ||
186         x1+thick/2 > image->width || x2+thick/2 > image->width ||
y1+thick/2 > image->height ||
187         y2+thick/2 > image->height)
188         return;
189     if (abs(y2-y1) > abs(x2-x1))
190         for (int i = -thick/2; i < thick/2; i++)
191             drawLine(x1+i, y1, x2+i, y2, color, image);
192     else
193         for (int i = -thick/2; i < thick/2; i++)
194             drawLine(x1, y1+i, x2, y2+i, color, image);
195 }
196
197 void gradient(png_bytep color, int width, Png *image) {
198     if (width <= 0 || !color || width >= image->height ||
199         width >= image->width) {
200         printf("Неверные аргументы для создания рамки.\n");
201         return;
202     }
203
204     for (int y = 0; y < width; y++)
205         for (int x = 0; x < image->width; x++) {
206             double pos = (double)y/image->height;
207             color[0] = pos * 255;
208             setPixel(&(image->row_pointers[y][x*4]), color);
209         }

```

```

210
211     for (int y = width; y < image->height; y++)
212         for (int x = 0; x < width; x++) {
213             double pos = (double)y/image->height;
214             color[0] = pos * 255;
215             setPixel(&(image->row_pointers[y][x*4]), color);
216         }
217
218     for (int y = image->height-width; y < image->height; y++)
219         for (int x = width; x < image->width; x++) {
220             double pos = (double)y/image->height;
221             color[0] = pos * 255;
222             setPixel(&(image->row_pointers[y][x*4]), color);
223         }
224
225     for (int y = width; y < image->height-width; y++)
226         for (int x = image->width-width; x < image->width; x++) {
227             double pos = (double)y/image->height;
228             color[0] = pos * 255;
229             setPixel(&(image->row_pointers[y][x*4]), color);
230         }
231     }
232 }
233
234 void frameLine(png_bytep color, int width, Png *image) {
235     if (width <= 0 || !color || width >= image->height ||
236         width >= image->width) {
237         printf("Неверные аргументы для создания рамки.\n");
238         return;
239     }
240
241     for (int y = 0; y < width; y++)
242         for (int x = 0; x < image->width; x++) {
243             if (y % 2)
244                 setPixel(&(image->row_pointers[y][x*4]), color);
245             if (x % 2) {
246                 double pos = (double) y / image->height;
247                 color[0] = pos * 255;
248                 setPixel(&(image->row_pointers[y][x*4]), color);
249             }
250         }
251
252     for (int y = width; y < image->height; y++)
253         for (int x = 0; x < width; x++) {
254             if (y % 2)
255                 setPixel(&(image->row_pointers[y][x*4]), color);
256             if (x % 2) {
257                 double pos = (double) y / image->height;
258                 color[0] = pos * 255;
259                 setPixel(&(image->row_pointers[y][x*4]), color);
260             }
261         }
262
263     for (int y = image->height-width; y < image->height; y++)
264         for (int x = width; x < image->width; x++) {
265             if (y % 2)
266                 setPixel(&(image->row_pointers[y][x*4]), color);
267             if (x % 2) {
268                 double pos = (double) y / image->height;
269                 color[0] = pos * 255;
270                 setPixel(&(image->row_pointers[y][x*4]), color);
271             }
272         }
273

```

```

274     for (int y = width; y < image->height-width; y++)
275         for (int x = image->width-width; x < image->width; x++) {
276             if (y % 2)
277                 setPixel(&(image->row_pointers[y][x*4]), color);
278             if (x % 2) {
279                 double pos = (double) y / image->height;
280                 color[0] = pos * 255;
281                 setPixel(&(image->row_pointers[y][x*4]), color);
282             }
283         }
284 }
285
286 void rectangles(int thick, png_bytep colorTo, png_bytep colorFrom,
287 Png* image) {
288     if (!image || !colorTo || !colorFrom || thick <= 0 || thick >=
289 10) {
290         printf("Неверные аргументы для обводки прямоугольников.\n");
291         return;
292     }
293
294     int height = image->height;
295     int width = image->width;
296     int count = 0;
297
298     Rectangle_t *rectangles = malloc(sizeof(Rectangle_t));
299
300     int **array = calloc(height, sizeof(int*));
301     for (int i = 0; i < height; i++)
302         array[i] = calloc(width, sizeof(int));
303
304     for (int row = 0; row < height; row++)
305         for (int col = 0; col < width; col++)
306             for (int r = row; r < height && isMatch(&(image->row_pointers[r][col*4]), colorFrom); r++)
307                 array[r][col] = 1;
308
309     for (int i = 0; i < height; i++)
310         for (int j = 0; j < width; j++)
311             if (array[i][j] == IN_RECTANGLE) {
312                 int flag = 0;
313                 int y = i, x = j;
314
315                 while (array[y][x] == IN_RECTANGLE && x+1 < width) {
316                     if (array[y][x+1] == CHECKED) flag = 1;
317                     array[y][x++] = CHECKED;
318                 }
319
320                 if (flag) continue;
321                 x--;
322                 y++;
323                 if (y >= height) continue;
324
325                 while (array[y][x] == IN_RECTANGLE && y+1 < height) {
326                     if (array[y+1][x] == CHECKED) flag = 1;
327                     array[y++][x] = CHECKED;
328                 }
329                 if (flag) continue;
330
331                 rectangles[count].x1 = j, rectangles[count].y1 = i;
332                 rectangles[count].x2 = x, rectangles[count].y2 =
333 y;
334                 rectangles = realloc(rectangles, (count+1) *
335 sizeof(Rectangle_t));

```

```

332     }
333
334     for (int i = 0; i < count; i++) {
335         drawLineThick(rectangles[i].x1, rectangles[i].y1,
336                       rectangles[i].x2, rectangles[i].y1, thick,
colorTo, image);
337         drawLineThick(rectangles[i].x1, rectangles[i].y1,
338                       rectangles[i].x1, rectangles[i].y2, thick,
colorTo, image);
339         drawLineThick(rectangles[i].x1, rectangles[i].y2,
340                       rectangles[i].x2, rectangles[i].y2, thick,
colorTo, image);
341         drawLineThick(rectangles[i].x2, rectangles[i].y1,
342                       rectangles[i].x2, rectangles[i].y2, thick,
colorTo, image);
343     }
344
345     for (int i = 0; i < height; i++)
346         free(array[i]);
347     free(array);
348     free(rectangles);
349 }
350
351 void printInfo(Png *image) {
352     if (!image)
353         return;
354     printf("
355           "Информация о считанном файле: \n"
356           "Тип файла: PNG изображение\n"
357           "Ширина картинки(в пикселях): %d\n"
358           "Высота картинки(в пикселях): %d\n"
359           "Глубина цвета: %d\n", image->width, image->height,
image->bit_depth);
360 }

```

/include/png_image.h

```

1  #ifndef PNG_IMAGE_H
2  #define PNG_IMAGE_H
3
4  #include <png.h>
5  #include <stdlib.h>
6  #include "structures.h"
7
8  #define GRADIENT 1
9  #define LINES 2
10 #define IN_RECTANGLE 1
11 #define CHECKED 2
12
13 typedef struct {
14     int width, height;
15     png_byte color_type;
16     png_byte bit_depth;
17
18     png_structp png_ptr;
19     png_infop info_ptr;
20     int number_of_passes;
21     png_bytep *row_pointers;
22 } Png;
23
24 Png* readPng(const char*);
25 int isMatch(png_bytep, png_bytep);
26 void changeColor(png_bytep, png_bytep, Png*);
27 void printInfo(Png*);
28 void rectangles(int, png_bytep, png_bytep, Png*);

```



```

29 void setPixel(png_bytep, png_bytep);
30 void drawLine(int, int, int, int, png_bytep, Png*);
31 void drawLineThick(int, int, int, int, int, png_bytep, Png*);
32 void writePng(const char*, Png*);
33 void frameLine(png_bytep, int, Png*);
34 void gradient(png_bytep, int, Png*);
35
36 #endif
/include/structures.h

1 #ifndef STRUCTURES_H
2 #define STRUCTURES_H
3
4 typedef struct {
5     int width;
6     int repaint;
7     int info;
8     int frame;
9     int color;
10    int to;
11    int frameType;
12    int rect;
13    int thick;
14    png_bytep frameColor;
15    png_bytep colorTo;
16    png_bytep colorFrom;
17    char *fileRead;
18    char *fileWrite;
19 } Opts;
20
21 typedef struct {
22     int x1;
23     int y1;
24     int x2;
25     int y2;
26 } Rectangle_t;
27
28 #endif

```