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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка PNG-файла

Студент гр. 1304 _____ Байков Е.С.

Преподаватель _____ Чайка К.В.

Санкт-Петербург

2022

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Байков Е.С.

Группа 1304

Тема работы: обработка PNG-файла

Исходные данные:

Реализация программы на языке Си, предоставляющую функционал по обработке PNG-файла. Программа должна иметь CLI или GUI.

Содержание пояснительной записки:

Перечисляются требуемые разделы пояснительной записки (обязательны разделы «Содержание», «Введение», «Заключение», «Список использованных источников»)

Предполагаемый объем пояснительной записки:

Не менее 30 страниц.

Дата выдачи задания: 22.03.2022

Дата сдачи реферата: 01.06.2022

Дата защиты реферата: 03.06.2022

Студент _____ Байков Е.С.

Преподаватель _____ Чайка К.В.

АННОТАЦИЯ

В ходе курсовой работы была реализована программа, вызываемая через командную строку с набором коротких или длинных ключей. Каждый ключ отвечает за функцию обработки изображения, путь до которого передается последним параметром. Также программа предполагает вывод справку о ключах. Программа работает лишь с изображениями формата PNG.

Функционал:

- Копирование области
- Замена цвета на другой
- Рисовании рамки для изображения
- Поиск прямоугольников заданного цвета и создание рамки вокруг каждого

SUMMARY

During the course work, a program was implemented, called via the command line with a set of short or long keys. Each key is responsible for the image processing function, the path to which is passed by the last parameter. The program also assumes the output of a certificate about the keys. The program only works with PNG images.

Functional:

- Copying an area
- Replacing a color with another one
- Drawing a frame for an image
- Search for rectangles of a given color and create a frame around each

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной курсовой работы является изучение методов обработки изображений в формате PNG при помощи соответствующей библиотеки libpng и при помощи ввода ключей в командную строку.

Программа должна быть вызвана из командной строки, обрабатывая поданные ей ключи и значения и обрабатывать изображение с помощью выбранной пользователем функции.

Для достижение поставленной цели требуется решить такие задачи, как:

- Изучение работы с getopt
- Изучение работы с libpng
- Понимание считывания и записи изображения
- Разработка опций по обработке изображения
- Тестирование разработанной программы

1. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

1.1. Используемые библиотеки

В данной курсовой работе использовались следующие библиотеки:

- stdio.h
- stdlib.h
- string.h
- setjmp.h
- getopt.h
- png.h
- ctype.h
- unistd.h

1.2. Работа с ключами при помощи getopt

Библиотека getopt была создана для разработки программ, использующих CLI. Библиотека реализует работу с ключами программы командной строки.

1.3. Работа с изображениями

Изображения хранятся в памяти в виде заголовка, описывающий структуру файла, и массива данных. Формат PNG использует сжатие, поэтому для корректной работы с данными таких файлов потребуется библиотека libpng.

После считывания изображения его обработка заключается в изменении двумерного массива, каждый элемент которого отвечает за цвет пикселя.

2. РЕАЛИЗАЦИЯ ПРОГРАММЫ

2.1. Функция main

В функции main задается набор ключей и обработчик для введённых пользователем опций. Также проверяются определенные параметры ввода (например, наличие названия обрабатываемого файла). Если пользователь вводит некорректную опцию или ничего не вводит, то выводиться сообщение об ошибке. Если же все введено корректно, то считывается файл, которы будет редактироваться, а затем с помощью конструкции switch-case обрабатываются ключи, введенные пользователем.

2.2. Функция копирования области

Функция копирования обрабатывает ключи, аргументами которых являются строки вида <число,число>, затем создается специальный буфер хранящий в себе нулевые значения, размер данного буфера определяется размером прямоугольника выделяемой области. По пикселю в буфер сохраняются значения его цветовых каналов, а после копирования каждый элемент буфера вставляется в прямоугольник такого же размера, но координата его верхнего левого угла другая. Затем буфер очищается, а изображение перезаписывается.

2.3. Функция смены цвета

Функция принимает два аргумента, каждый из которых может быть либо строкой с названием цвета (green, blue), либо в виде кода rgb(a) поданном в формате <число,число,число>, либо для RGBA картинки - <число,число,число,число>. Затем создаются два массива, которые заполняются соответствующими компонентами, а после все пиксели в изображении цвета пикселей сравниваются с данными в одном массиве и заменяются на данные в другом. После данные массивов очищаются.

2.4. Функция рисования рамки вокруг изображения

Пользователь выбирает рамку, ее цвет и ширину. В зависимости от выбранного типа рамки может быть нарисована самая обычная рамка, рамка в

виде ковра Сперинского, шахматной доски либо в виде туннеля. Для рисования ковра Серпинского использовалась рекурсивная формула, сохраняющая в буфер пиксели закрашенного цвета и определяющая размер квадратов в зависимости от ширины. В функции рисования рамки в виде тоннеля использовался алгоритм Брезенхема.

2.5. Функция поиска прямоугольников

Функция пробегается по пикселям картинки и находит первый пиксель заданного цвета. Затем «очерчивает» границу прямоугольника и проверяет его на заполненность. Также проверяется и выступающие пиксели. Если прямоугольник найден, то координаты его верхнего левого угла, а также длина и ширина записываются в соответствующую структуру, предварительно создан массив таких структур. Затем с помощью цикла очерчивается рамка вокруг прямоугольника.

3. ТЕСТИРОВАНИЕ

3.1. Тестирование функции копирования области

- Входные данные: ./PNG.out

Выходные данные: **<краткая_справка>**

- Входные данные: ./PNG.out -h

Выходные данные: **<справка>**

- Входные данные: ./PNG.out -c

Выходные данные: **-? -h --help**

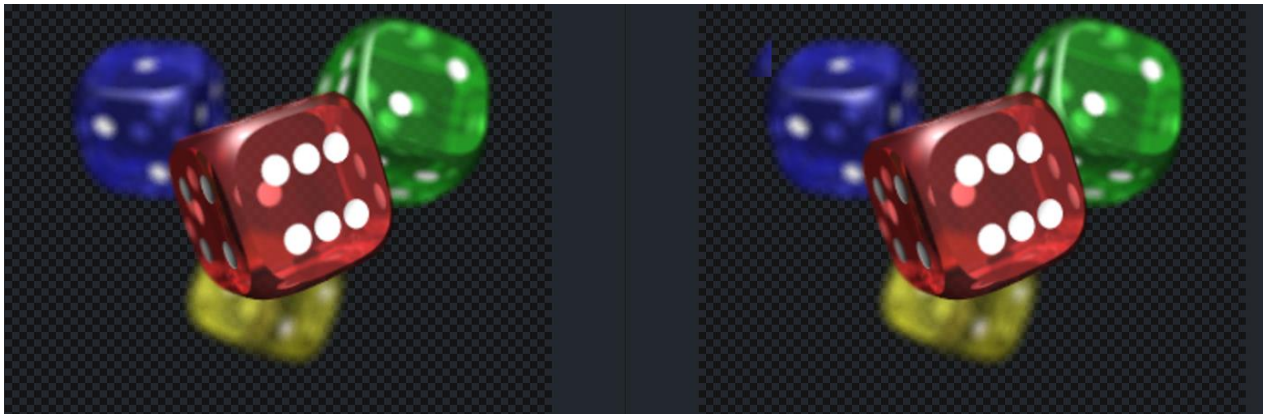
There is no any PNG files

- Входные данные: ./PNG.out -c new.png

Выходные данные: **You did not choose one or more options of copy command** <краткая_справка>

- Входные данные: ./PNG.out -c -l 20,20 -r 100,100 -e 0,0 test.png

Выходные данные:



3.2. Тестирование функции смены цвета

- Входные данные: ./PNG.out -t

Выходные данные: **-? -h --help**

There is no any PNG files

- Входные данные: ./PNG.out -t new.png

Выходные данные: **You did not choose one or more options of copy command** <краткая_справка>

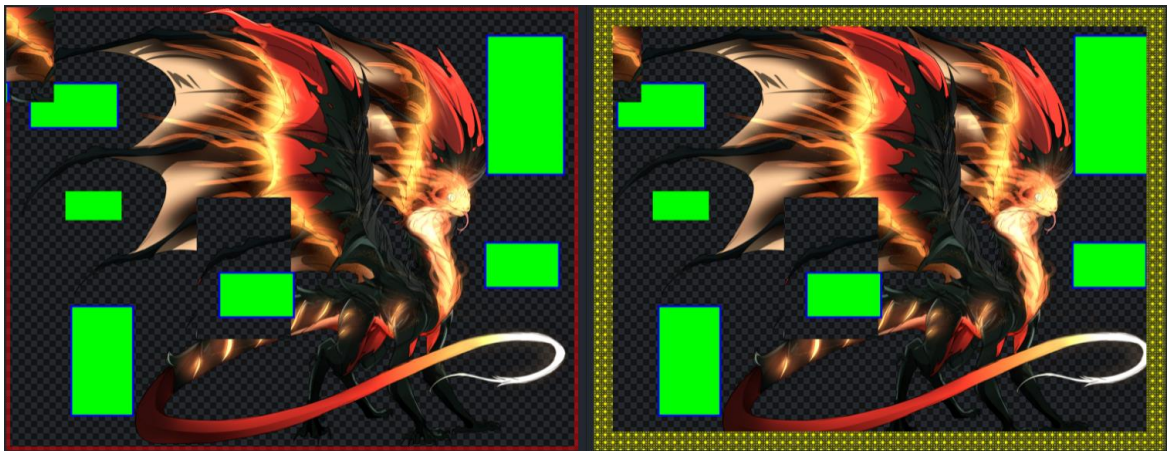
- Входные данные: ./PNG.out -t -f red -t green 2drp.png

Выходные данные:

3.3. Тестирование функции рисования рамки

- Входные данные: ./PNG.out -f -t fractal -c yellow --width 200 2drp.png

Выходные данные:



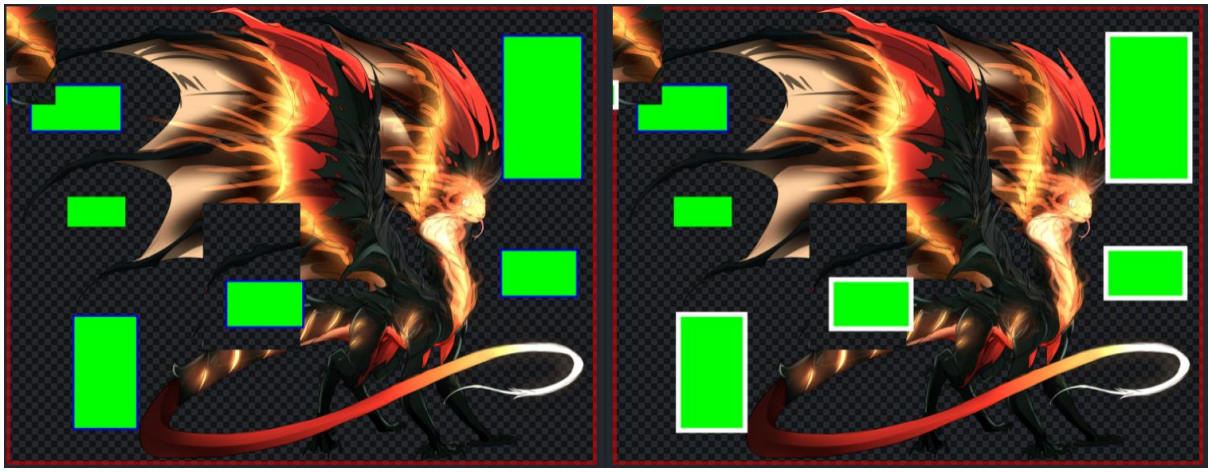
- Входные данные: ./PNG.out -f -t tunnel -c yellow --width 200 2drp.png

Выходные данные:

3.4. Тестирование функции поиска прямоугольников

- Входные данные: ./PNG.out -r -c green --framecolor white -w 20 2drp.png

Выходные данные:



ЗАКЛЮЧЕНИЕ

В ходе данной курсовой работы была реализована программа для обработки PNG-изображения с помощью ключей, подаваемых пользователем, а также изучены принципы работы с getopt и библиотеки libpng.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. http://cppstudio.com/
2. https://stackoverflow.com/
3. https://www.cplusplus.com/
4. http://www.libpng.org/pub/png/libpng-1.2.5-manual.html

ПРИЛОЖЕНИЕ А
КОД ПРОГРАММЫ

- **Файл main.c**

```
#include "info.h"

int main(int argc, char **argv){
   const char *opts = "ctfrqhs?";
   char file_name[50];
   if(argc == 1){
      printHelp(argc);
      exit(0);
   }
   else if(argc == 2 && (!strcmp("-h", argv[argc-1]) || !strcmp("--help", argv[argc-
1])))){
      printHelp(0);
      exit(0);
   }
   else if (!strstr(argv[argc-1], ".png")){
      printHelp(6);
      exit(0);
   }
   else if(argc >= 3){
      strcpy(file_name, argv[--argc]);
   }
   sPng image;
   struct option longOpt[] = {
      {"copy", no_argument, NULL, 'c'},
      {"transform", no_argument, NULL, 't'},
      {"frame", no_argument, NULL, 'f'},
      {"rects", no_argument, NULL, 'r'},
      {"standart", no_argument, NULL, 's'},
      {"help", no_argument, NULL, 'h'},
      {"quit", no_argument, NULL, 'q'},
      {NULL, 0, NULL, 0}
   };
   if(!readPNG(file_name, &image)){
```

```
        puts("Wrong");
        exit(0);
    }
    int opt, opt_index;
    while((opt = getopt_long(argc, argv, opts, longOpt, &opt_index)) != -1){

        switch(opt){
            case 'c':
                copyAreaGet(argc, argv, &opt_index, &image, file_name);
                break;
            case 't':
                transformGet(argc, argv, &opt_index, &image, file_name);
                break;
            case 'f':
                makeFrameGet(argc, argv, &opt_index, &image, file_name);
                break;
            case 'r':
                rectsGet(argc, argv, &opt_index, &image, file_name);
                break;
            case 'h':
                printHelp(0);
                break;
            case 'q':
                exit(0);
                break;
            case 's':
                info(&image);
                break;
            case '?':
            default:
                printHelp(0);
                break;
        }
    }
    return 0;
}
```

- **Файл copy.c**

```
#include "info.h"
```

```
void copyAreaPNG(sPng *image, char *left_up, char *right_down, char
*destination){
    int color_indicator = checkColorType(image);
    if(color_indicator == -1) return;

    char *start = (char *)calloc(strlen(left_up) + 1, sizeof(char));
    char *end = (char *)calloc(strlen(right_down) + 1, sizeof(char));
    char *dest = (char *)calloc(strlen(destination) + 1, sizeof(char));
    strcpy(start, left_up);
    strcpy(end, right_down);
    strcpy(dest, destination);
    int startx, starty, endx, endy, destx, desty;
    char *ptr1 = strtok(start, ","), *ptr2 = strtok(NULL, ",");

    if(isNum(ptr1) && isNum(ptr2)){
        startx = atoi(ptr1);
        starty = atoi(ptr2);
    }
    else{
        puts("Start point has wrong arguments");
        free(start);
        free(end);
        free(dest);
        return;
    }
    ptr1 = strtok(end, ","), ptr2 = strtok(NULL, ",");
    if(isNum(ptr1) && isNum(ptr2)){
        endx = atoi(ptr1);
        endy = atoi(ptr2);
    }
    else{
        puts("End point has wrong arguments");
        free(start);
        free(end);
        free(dest);
        return;
    }
```

```
ptr1 = strtok(dest, ","), ptr2 = strtok(NULL, ",");
if(isNum(ptr1) && isNum(ptr2)){
   destx = atoi(ptr1);
   desty = atoi(ptr2);
}
else{
   puts("Destination point has wrong arguments");
   free(start);
   free(end);
   free(dest);
   return;
}

free(start);
free(end);
free(dest);

if(starty < 0 || starty >= image->height || startx < 0 || startx >= image->width){
   puts("Wrong position start");
   return;
}

if(endy < 0 || endy >= image->height || endx < 0 || endx >= image->width){
   puts("Wrong position end");
   return;
}

if(desty < 0 || desty >= image->height || destx < 0 || destx >= image->width){
   puts("Wrong position destination");
   return;
}

if(startx >= endx || starty >= endy){
   puts("Start positions bigger then end ones");
   puts("   Start pos    End pos");
   if(startx >= endx) printf("x: %d    >      %d\n", startx, endx);
   if(starty >= endy) printf("y: %d    >      %d\n", starty, endy);
   return;
}
```

```
    png_bytep *buf = (png_bytep *)calloc(endy - starty,sizeof(png_bytep));
    for(int i = 0; i < endy - starty; i++)
        buf[i] = (png_byte *)calloc(1, png_get_rowbytes(image->png_ptr,
image->info_ptr));

    for(int i = starty; i < image->height && i < endy; i++){
        png_byte *row = image->row_pointers[i];
        png_byte *bufr = buf[i-starty];
        for(int j = startx; j < image->width && j < endx; j++){
            png_byte *ptr = &(row[j * color_indicator]);
            png_byte *buf_ptr = &(bufr[(j - startx) * color_indicator]);
            buf_ptr[0] = ptr[0];
            buf_ptr[1] = ptr[1];
            buf_ptr[2] = ptr[2];

            if(color_indicator == 4) buf_ptr[3] = ptr[3];

        }
    }
    for(int i = desty; i < image->height && i < endy - starty + desty; i++){
        png_byte *row = image->row_pointers[i];
        png_byte *bufr = buf[i-desty];
        for(int j = destx; j < image->width && j < endx - startx + destx; j++){
            png_byte *ptr = &(row[j*color_indicator]);
            png_byte *buf_ptr = &(bufr[(j-destx) * color_indicator]);
            ptr[0] = buf_ptr[0];
            ptr[1] = buf_ptr[1];
            ptr[2] = buf_ptr[2];

            if(color_indicator == 4)ptr[3] = buf_ptr[3];

        }
    }
    for(int i = 0; i < endy - starty; i++){
        free(buf[i]);
    }
    free(buf);
}
```

```
void copyAreaGet(int argc, char **argv, int *opt_index, sPng *image, char
*filename){
   const char *opts = "l:r:e:?";
   struct option lOpt[] = {
      {"leftup", required_argument, NULL, 'l'},
      {"rightlow", required_argument, NULL, 'r'},
      {"endleftup", required_argument, NULL, 'e'},
      {NULL, 0, NULL, 0}
   };
   int opt;
   char *start = NULL, *end = NULL, *dest = NULL;
   while(-1 != (opt = getopt_long(argc, argv, opts, lOpt, opt_index))){
      switch (opt)
      {
         case 'l':
            start = optarg;
            break;
         case 'r':
            end = optarg;
            break;
         case 'e':
            dest = optarg;
            break;
         case '?':
         default:
            printHelp(2);
            writePNG(filename, image);
            exit(0);
            break;
      }
   }
   if(start == NULL || end == NULL || dest == NULL){
      puts("You did not choose one or more options of copy command");
      printHelp(5);
      writePNG(filename, image);
      exit(0);
   }
   copyAreaPNG(image, start, end, dest);
```

```
    if(!writePNG(filename, image)){
        exit(0);
    }
}
```

- **Файл help.c**

```
#include "info.h"

int isNum(char *str){
    int len = strlen(str);
    for(int i = 0; i < len; i++){
        if(!isdigit(str[i]))
            return 0;
    }
    return 1;
}

int setParams(int *parametrs, int ind, char *colour){
    char *color = (char *)calloc(strlen(colour) + 1, sizeof(char));
    if(!color) return 0;
    strcpy(color, colour);
    int i = 0, num;
    char *ptr;
    parametrs[3] = 255;
    switch (ind)
    {
        case -1:
            i = 0;
            ptr = strtok(color, ",");
            while(ptr){
                if(!isNum(ptr)){
                    free(color);
                    puts("Wrong color");
                    return 0;
                }
                num = atoi(ptr);
                if(num > 255 || num < 0){
                    free(color);
                    puts("Wrong argument of RGB/RGBA code");
                    printf("%d %c %d\n", num, num>255?'>':'<', num>255?255:0);
```

```
            return 0;

        }
        parametrs[i++] = num;
        ptr = strtok(NULL, ",");
    }
    break;
case 0:
    parametrs[0] = 255;
    break;
case 1:
    parametrs[1] = 255;
    break;
case 2:
    parametrs[2] = 255;
    break;
case 3:
    parametrs[0] = 150;
    parametrs[1] = 150;
    parametrs[2] = 150;
    break;
case 4:
    parametrs[0] = 255;
    parametrs[1] = 255;
    break;
case 5:
    parametrs[0] = 160;
    parametrs[1] = 32;
    parametrs[2] = 240;
    break;
case 6:
    parametrs[0] = 255;
    parametrs[1] = 255;
    parametrs[2] = 255;
    break;
case 7:
    parametrs[0] = 0;
    parametrs[1] = 0;
    parametrs[2] = 0;
```

```
            break;
        case 8:
            parametrs[0] = 0;
            parametrs[1] = 255;
            parametrs[2] = 255;
            break;
        case 9:
            parametrs[0] = 255;
            parametrs[1] = 165;
            break;
        default:
            break;
    }
    free(color);
    return 1;
}

void printHelp(int arg){
    puts("-? -h --help\n");
    if(arg == 1 || !arg){
        puts("The program has the following structure:\n\t<name_of_executable_file>
<names_of_options> <file_name>");
        puts("\tFile must have png format -> <file_name.png>");
        puts("-s --standart - to learn standart colors that are used in the function
transform and the frames that are used in the function frame");
        puts("If you want to use all functions correctly write the file name in the end
of your commands");
        puts("If you want to enter the code of color in RGB or RGBA the structure
must be
<Red_Component>,<Green_Component>,<Blue_Component>,<Alpha_Compone
nt>\n(The last one uses if you have RGBA picture)");
        puts("");
    }
    if(arg == 2 || !arg){
        puts("-c --copy - copy pixels from one area to another");
        puts("\t-l --leftup <x_axis>,<y_axis> - coordinates of upper-left corner");
        puts("\t-r --rightlow <x_axis>,<y_axis> - coordinates of lower-right corner");
        puts("\t-e --endleftup <x_axis>,<y_axis> - coordinates of the upper-left corner
where the program paste copied area");
```

```
        puts("");
        puts("WARNING: y axis directed from top to bottom\nupper-left corner has
coordinates (0,0) and lower-left corner has coordinates (0, height of picture - 1)");
        puts("");
    }
    if(arg == 3 || !arg){
        puts("-t --transform - change one color to another in a given area");
        puts("\t -f --fromcolor <value>- the color to change");
        puts("\t -t --tocolor <value> - the color to change the current color");
        puts("");
    }
    if(arg == 4 || !arg){
        puts("-f --frame - make a frame aroud the picture");
        puts("\t -t --type <value> - type of pattern");
        puts("\t -c --color <value> - color of pattern");
        puts("\t -w --width <value> - width of pattern");
        puts("");
    }

    if(arg == 5 || !arg){
        puts("-r --rects - find rectangles and make the frame around them");
        puts("\t -c --color <value> - color of rectanges");
        puts("\t -f --framecolor <value> - color of frame");
        puts("\t -w --width <value> - width of frame");
        puts("");
    }
    if(arg == 6){
        puts("There is no any PNG files");
        puts("");
    }
}

int readPNG(char *filename, sPng *image){
    FILE *fp = fopen(filename, "rb");
    char header[8];
    if(!fp) {
        puts("Could not read file");
        return 0;
    };
```

```
   fread(header, 1, 8, fp);

   if(png_sig_cmp(header, 0, 8)) {
      fclose(fp);
      puts("Could not reead header");
      return 0;
   }

   image->png_ptr = png_create_read_struct(PNG_LIBPNG_VER_STRING,
NULL, NULL, NULL);
   if(!image->png_ptr) {
      fclose(fp);
      puts("Could not create read struct");
      return 0;
   }

   image->info_ptr = png_create_info_struct(image->png_ptr);
   if(!image->info_ptr) {
      png_destroy_read_struct(&image->png_ptr, (png_infopp)NULL,
(png_infopp)NULL);
      fclose(fp);
      puts("Could not create info struct");
      return 0;
   }

   if(setjmp(png_jmpbuf(image->png_ptr))) {
      png_destroy_read_struct(&image->png_ptr, &image->info_ptr,
(png_infopp)NULL);
      fclose(fp);
      puts("Something went wrong");
      return 0;
   }

   png_init_io(image->png_ptr, fp);
   png_set_sig_bytes(image->png_ptr, 8);
   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);
    image->number_of_passes = png_set_interlace_handling(image->png_ptr);
    png_read_update_info(image->png_ptr, image->info_ptr);

    if(setjmp(png_jmpbuf(image->png_ptr))) {
        png_destroy_read_struct(&image->png_ptr, &image->info_ptr,
(png_infopp)NULL);
        fclose(fp);
        puts("Something went wrong");
        return 0;
    }

    image->row_pointers = (png_bytep *)calloc(image->height, sizeof(png_bytep));
    for(int i = 0; i < image->height; i++){
        image->row_pointers[i] = (png_byte
*)malloc(png_get_rowbytes(image->png_ptr, image->info_ptr));
    }

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

    return 1;
    fclose(fp);
}

int writePNG(char *filename, sPng *image){
    FILE *fout = fopen(filename, "wb");

    if(!fout){
        puts("Could not read file");
        return 0;
    }

    image->png_ptr = png_create_write_struct(PNG_LIBPNG_VER_STRING,
NULL, NULL, NULL);

    if(!image->png_ptr){
        fclose(fout);
        png_destroy_write_struct(&image->png_ptr, (png_infopp)NULL);
        return 0;
```

```
    }

    image->info_ptr = png_create_info_struct(image->png_ptr);

    png_init_io(image->png_ptr, fout);

    if(setjmp(png_jmpbuf(image->png_ptr))){
        fclose(fout);
        png_destroy_write_struct(&image->png_ptr, &image->info_ptr);
        return 0;
      }

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

    if(setjmp(png_jmpbuf(image->png_ptr))){
      fclose(fout);
      png_destroy_write_struct(&image->png_ptr, &image->info_ptr);
      return 0;
    }

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

    if(setjmp(png_jmpbuf(image->png_ptr))){
      fclose(fout);
      png_destroy_write_struct(&image->png_ptr, &image->info_ptr);
      return 0;
    }

    png_write_end(image->png_ptr, NULL);
    for(int i = 0; i < image->height; i++){
      free(image->row_pointers[i]);
    }
    free(image->row_pointers);
    fclose(fout);
}
```

```
int checkColorType(sPng *image){
   switch (png_get_color_type(image->png_ptr, image->info_ptr))
   {
      case PNG_COLOR_TYPE_PALETTE:
         puts("Your type is PALETTE, but expected RGB or RGBA"); // if the
picture is not RGB or RGBA drop the program
         return -1;
         break;
      case PNG_COLOR_TYPE_GA:
         puts("Your type is GA, but expected RGB or RGBA");
         return -1;
         break;
      case PNG_COLOR_TYPE_GRAY:
         puts("Your type is GRAY, but expected RGB or RGBA");
         return -1;
         break;
      case PNG_COLOR_TYPE_RGB:
         return 3;
         break;
      case PNG_COLOR_TYPE_RGBA:
         return 4;
         break;
      default:
         break;
   }
}

void info(sPng *image){
   puts("Standart colors");

puts("\t*black\n\t*yellow\n\t*orange\n\t*white\n\t*green\n\t*blue\n\t*cyan\n\t*gra
y\n\t*red\n\t*purple");
   puts("");
   puts("Standart frames");
   puts("\t*common\n\t*fractal\n\t*tunnel\n\t*chess");
   int color;
   switch (checkColorType(image))
   {
   case -1:
```

```
        exit(0);
        break;
    case 3:
        color = 3;
        break;
    case 4:
        color = 4;
        break;
    default:
        break;
    }
    printf("Color type - %s\n", color == 3 ? "RGB" : "RGBA");
    printf("Color depth - %d\n", image->bit_depth);
    printf("Height of the picture - %d\n", image->height);
    printf("Width of the picture - %d\n", image->width);
}
```

- **Файл transform.c**

```
#include "info.h"

void transformGet(int argc, char **argv, int *opt_index, sPng *image, char
*filename){
    const char *opts = "f:t:?";
    struct option lOpt[] = {
        {"fromcolor", required_argument, NULL, 'f'},
        {"tocolor", required_argument, NULL, 't'},
        {NULL, 0, NULL, 0}
    };
    char *color1 = NULL, *color2 = NULL;
    int opt;
    while(-1 != (opt = getopt_long(argc, argv, opts, lOpt, opt_index))){
        switch (opt)
        {
            case 'f':
                color1 = optarg;
                break;
            case 't':
                color2 = optarg;
                break;
```

```
            case '?':
            default:
                printHelp(3);
                writePNG(filename, image);
                exit(0);
                break;
        }
    }
    if(color1 == NULL || color2 == NULL){
        puts("You did not choose one or more options of transform command");
        printHelp(5);
        writePNG(filename, image);
        exit(0);
    }
    transformPNG(image, color1, color2);
    if(!writePNG(filename, image)){
        exit(0);
    }
}

void transformPNG(sPng *image, char *color1, char *color2){
    int color_indicator = checkColorType(image);
    if(color_indicator == -1) return;

    char colours[10][20] = {"red", "green", "blue", "gray", "yellow", "purple",
"white", "black", "cyan", "orange"};
    int index1 = -1, index2 = -1;

    for(int i = 0; i < 10; i++){
        if(!strcmp(color1, colours[i])) index1 = i;
        if(!strcmp(color2, colours[i])) index2 = i;
    }
    int *parametrs1 = (int *)calloc(4, sizeof(int));
    int *parametrs2 = (int *)calloc(4, sizeof(int));

    if(!setParams(parametrs1, index1, color1) || !setParams(parametrs2, index2,
color2)){
        free(parametrs1);
        free(parametrs2);
```

```
        return;
    }


    for(int i = 0; i < image->height; i++){
        png_byte *row = image->row_pointers[i];
        for(int j = 0; j < image->width; j++){
            png_byte *ptr = &(row[j * color_indicator]);
            if(ptr[0] == parametrs1[0] && ptr[1] == parametrs1[1] && ptr[2] ==
parametrs1[2]){
                ptr[0] = parametrs2[0];
                ptr[1] = parametrs2[1];
                ptr[2] = parametrs2[2];
            }
        }
    }
    free(parametrs1);
    free(parametrs2);
}
```

- **Файл rects.c**

```
#include "info.h"


int isRectColor(sPng *image, int x, int y, int indicator, int *params){
    if(x < 0 || y < 0) return 0;
    if(x >= image->width || y >= image->height) return 0;
    png_byte *row = image->row_pointers[y];
    png_byte *ptr = &(row[x*indicator]);
    for(int i = 0; i < indicator; i++){
        if(ptr[i] != params[i]) return 0;
    }
    return 1;
}

int isRectFill(sPng *image, int width, int height, int indicator, int *params, int x,
int y){
    for(int i = y; i < y + height; i++){
        for(int j = x; j < x + width; j++){
            if(!isRectColor(image, j, i, indicator, params)) return 0;
```

```
        }
    }
    return 1;
}

int isRectShape(sPng *image, int width, int height, int indicator, int *params, int x,
int y){
    for(int j = x; j < x + width - 1; j++){
        if(isRectColor(image, j, y-1, indicator, params)) return 0;
        if(isRectColor(image, j, y+height, indicator, params)) return 0;
    }
    for(int i = y; i < y + height - 1; i++){
        if(isRectColor(image, x - 1, i, indicator, params)) return 0;
        if(isRectColor(image, x + width, i, indicator, params)) return 0;
    }
    return 1;
}

void drawRectFrame(sPng *image, Rect *arr, int *params, int width, int count, int
indicator){
    for(int n = 0; n < count; n++){
        int i;
        for(i = arr[n].y - width; i < arr[n].y; i++){
            if(i >= 0){
                png_byte *row = image->row_pointers[i];
                for(int j = arr[n].x - width; j < arr[n].x + arr[n].width + width && j <
image->width; j++){
                    if(j >= 0){
                        png_byte *ptr = &(row[j*indicator]);
                        ptr[0] = params[0];
                        ptr[1] = params[1];
                        ptr[2] = params[2];
                        if(indicator == 4) ptr[3] = params[3];
                    }
                }
            }
        }
        for(i; i < arr[n].y + arr[n].height; i++){
            if(i >= 0){
```

```
            png_byte *row = image->row_pointers[i];
            for(int j = arr[n].x - width; j < arr[n].x + arr[n].width + width && j <
image->width; j++){
                if(j >= 0 && (j < arr[n].x || j >= arr[n].x + arr[n].width)){
                    png_byte *ptr = &(row[j*indicator]);
                    ptr[0] = params[0];
                    ptr[1] = params[1];
                    ptr[2] = params[2];
                    if(indicator == 4) ptr[3] = params[3];
                }
            }
        }
    }

    for(i; i < arr[n].y + arr[n].height + width && i < image->height; i++){
        if(i >= 0){
            png_byte *row = image->row_pointers[i];
            for(int j = arr[n].x - width; j < arr[n].x + arr[n].width + width && j <
image->width; j++){
                if(j >= 0){
                    png_byte *ptr = &(row[j*indicator]);
                    ptr[0] = params[0];
                    ptr[1] = params[1];
                    ptr[2] = params[2];
                    if(indicator == 4) ptr[3] = params[3];
                }
            }
        }
    }
  }
}

void rectsPNG(sPng *image, char *rect_color, char *frame_color, char *width){
  int color_indicator = checkColorType(image);
  if(color_indicator == -1) return;

  char colours[10][20] = {"red", "green", "blue", "gray", "yellow", "purple",
"white", "black", "cyan", "orange"};
  int index_rect = -1, index_frame = -1;
```

```
    for(int i = 0; i < 10; i++){
        if(!strcmp(rect_color, colours[i])) index_rect = i;
        if(!strcmp(frame_color, colours[i])) index_frame = i;
    }


    int *rects_par = (int *)calloc(4, sizeof(int));
    int *frame_par = (int *)calloc(4, sizeof(int));
    if(!setParams(rects_par, index_rect, rect_color) || !setParams(frame_par,
index_frame, frame_color)){
        free(rects_par);
        free(frame_par);
        return;
    }

    int rect_count = 0;
    Rect *arr_rects = (Rect *)calloc(image->height / 2, sizeof(Rect));
    for(int i = 0; i < image->height - 1; i++){
        for(int j = 0; j < image->width - 1; j++){
            if(isRectColor(image, j, i, color_indicator, rects_par)
&& !isRectColor(image, j-1, i, color_indicator, rects_par) && !isRectColor(image,
j, i-1, color_indicator, rects_par)){
                int x = j, y = i;
                int width = 0, height = 0;
                while(isRectColor(image, x, i, color_indicator, rects_par)){
                    ++x;
                    ++width;
                }
                while(isRectColor(image, j, y, color_indicator, rects_par)){
                    ++y;
                    ++height;
                }
                if(isRectFill(image, width, height, color_indicator, rects_par, j, i)){
                    if(isRectShape(image, width, height, color_indicator, rects_par, j, i)){
                        arr_rects[rect_count].x = j;
                        arr_rects[rect_count].y = i;
                        arr_rects[rect_count].height = height;
                        arr_rects[rect_count].width = width;
```

```
                    ++rect_count;
                }
            }
        }
    }
    if(!rect_count){
        puts("There is not any rectangle of this color");
        free(rects_par);
        free(frame_par);
        free(arr_rects);
        return;
    }

    if(!isNum(width)){
        puts("Wrong width");
        free(rects_par);
        free(frame_par);
        free(arr_rects);
        return;
    }

    drawRectFrame(image, arr_rects, frame_par, atoi(width), rect_count,
color_indicator);
    free(frame_par);
    free(rects_par);
    free(arr_rects);
}

void rectsGet(int argc, char **argv, int *opt_index, sPng *image, char *filename){
    const char *opts = "c:f:w:?";
    struct option lOpt[] = {
        {"color", required_argument, NULL, 'c'},
        {"framecolor", required_argument, NULL, 'f'},
        {"width", required_argument, NULL, 'w'},
        {NULL, 0, NULL, 0}
    };
    char *rect_color = NULL, *frame_color = NULL, *width = NULL;
    int opt;
```

```
    while (-1 != (opt = getopt_long(argc, argv, opts, lOpt, opt_index)))
    {
        switch(opt){
            case 'c':
                rect_color = optarg;
                break;
            case 'f':
                frame_color = optarg;
                break;
            case 'w':
                width = optarg;
                break;
            case '?':
            default:
                printHelp(5);
                writePNG(filename, image);
                exit(0);
                break;
        }
    }
    if(rect_color == NULL || frame_color == NULL || width == NULL){
        puts("You did not choose one or more options of rectangle command");
        printHelp(5);
        writePNG(filename, image);
        exit(0);
    }
    rectsPNG(image, rect_color, frame_color, width);
    if(!writePNG(filename, image)){
        exit(0);
    }
}
```

- **Файл frames.c**

```
#include "info.h"

int findMaxDegree(int num){
    if(num < 1){
        return 0;
    }
```

```
    int x = 1;
    while(x <= num){
        x *= 3;
    }
    return x / 3;
}

void fillPixel(png_bytep *square, int x, int width, int *params, int indicator){
    for(int i = 0; i < width; i++){
        png_byte *row = square[i];
        for(int j = x; j < x + width; j++){
            png_byte *ptr = &(row[j * indicator]);
            ptr[0] = params[0];
            ptr[1] = params[1];
            ptr[2] = params[2];
            if(indicator == 4) ptr[3] = params[3];
        }
    }
}

void fillSquares(png_bytep *square, int gx, int width, int *params, int indicator){
    if (width == 1)
        return;
    int square_width = width / 3;
    for (int y = 0; y < width; y += square_width) {
        for (int x = 0; x < width; x += square_width) {
            if (x == y && x == square_width) {
                fillPixel(square + y, gx + x, square_width, params, indicator);
            } else {
                fillSquares(square + y, gx + x, square_width, params, indicator);
            }
        }
    }
}

png_bytep *drawSquare(int width, struct Png *image, int *params, int indicator){
    int maxDeg = findMaxDegree(width);

    png_bytep *buf = (png_bytep *)calloc(maxDeg, sizeof(png_bytep));
```

```
    for(int i = 0; i < maxDeg; i++)
        buf[i] = (png_byte *)calloc(1, png_get_rowbytes(image->png_ptr,
image->info_ptr));

    fillSquares(buf, 0, maxDeg, params, indicator);
    return buf;
}

void drawFractal(struct Png *image, char *color, char *width, int indicator){
    char colours[10][20] = {"red", "green", "blue", "gray", "yellow", "purple",
"white", "black", "cyan", "orange"};
    int ind = -1;
    for(int i = 0; i < 10; i++)
        if(!strcmp(color, colours[i])) ind = i;
    int *parametrs = (int *)calloc(4, sizeof(int));

    if(!setParams(parametrs, ind, color)){
        free(parametrs);
        return;
    }
    if(!isNum(width)) {
        free(parametrs);
        puts("Wrong width");
        return;
    }

    int i_width = atoi(width);

    if(i_width * 2 > image->height || i_width * 2 > image->width){
        free(parametrs);
        puts("Too big width");
        return;
    }

    int maxDeg = findMaxDegree(i_width);
    if(maxDeg == 0){
        free(parametrs);
        return;
    }
```

```
    png_bytep *square = drawSquare(i_width, image, parametrs, indicator);
    for (int i = 0; i < image->height; i++){
        png_byte *row = image->row_pointers[i];
        png_byte *sq_row = square[i % maxDeg];
        for(int j = 0; j < image->width; j++){
            png_byte *ptr = &(row[j*indicator]);
            png_byte *sq_ptr = &(sq_row[(j % maxDeg) * indicator]);
            if(i < i_width){
                ptr[0] = sq_ptr[0];
                ptr[1] = sq_ptr[1];
                ptr[2] = sq_ptr[2];
                if(indicator == 4) ptr[3] = sq_ptr[3];
            }
            else if(i >= i_width && i < image->height - i_width && (j < i_width || j >=
image->width - i_width)){
                ptr[0] = sq_ptr[0];
                ptr[1] = sq_ptr[1];
                ptr[2] = sq_ptr[2];
                if(indicator == 4) ptr[3] = sq_ptr[3];
            }
            else if(i >= image->height - i_width){
                ptr[0] = sq_ptr[0];
                ptr[1] = sq_ptr[1];
                ptr[2] = sq_ptr[2];
                if(indicator == 4) ptr[3] = sq_ptr[3];
            }
        }
    }


    free(square);
    free(parametrs);
}

void drawCommon(sPng *image, char *color, char *width, int indicator){
    char colours[10][20] = {"red", "green", "blue", "gray", "yellow", "purple",
"white", "black", "cyan", "orange"};
    int ind = -1;
    for(int i = 0; i < 10; i++)
```

```
    if(!strcmp(color, colours[i])) ind = i;
int *parametrs = (int *)calloc(4, sizeof(int));

if(!setParams(parametrs, ind, color)){
    free(parametrs);
    return;
}
if(!isNum(width)) {
    free(parametrs);
    puts("Wrong width");
    return;
}


int i_width = atoi(width);

if(i_width * 2 > image->height || i_width * 2 > image->width){
    free(parametrs);
    puts("Too big width");
    return;
}

int y;
printf("%d %d %d\n", parametrs[0], parametrs[1], parametrs[2]);
for(y = 0; y < i_width; y++){
    png_byte *row = image->row_pointers[y];
    for(int j = 0; j < image->width; j++){
        png_byte *ptr = &(row[j * indicator]);
        ptr[0] = parametrs[0];
        ptr[1] = parametrs[1];
        ptr[2] = parametrs[2];
        if(indicator == 4) ptr[3] = parametrs[3];
    }
}
for(y; y < image->height - i_width; y++){
    png_byte *row = image->row_pointers[y];
    for(int j = 0; j < i_width; j++){
        png_byte *ptr = &(row[j * indicator]);
        ptr[0] = parametrs[0];
```

```
            ptr[1] = parametrs[1];
            ptr[2] = parametrs[2];
            if(indicator == 4) ptr[3] = parametrs[3];
        }
        for(int j = image->width - i_width; j < image->width; j++){
            png_byte *ptr = &(row[j * indicator]);
            ptr[0] = parametrs[0];
            ptr[1] = parametrs[1];
            ptr[2] = parametrs[2];
            if(indicator == 4) ptr[3] = parametrs[3];
        }
    }
    for(y; y < image->height; y++){
        png_byte *row = image->row_pointers[y];
        for(int j = 0; j < image->width; j++){
            png_byte *ptr = &(row[j * indicator]);
            ptr[0] = parametrs[0];
            ptr[1] = parametrs[1];
            ptr[2] = parametrs[2];
            if(indicator == 4) ptr[3] = parametrs[3];
        }
    }
    free(parametrs);
}

void drawChess(sPng *image, char *color, char *width, int indicator){
    char colours[10][20] = {"red", "green", "blue", "gray", "yellow", "purple",
"white", "black", "cyan", "orange"};
    int ind = -1;
    for(int i = 0; i < 10; i++)
        if(!strcmp(color, colours[i])) ind = i;
    int *parametrs = (int *)calloc(4, sizeof(int));

    if(!setParams(parametrs, ind, color)){
        free(parametrs);
        return;
    }
    if(!isNum(width)) {
        free(parametrs);
```

```
    puts("Wrong width");
    return;
}
int i_width = atoi(width);

if(i_width * 2 > image->height || i_width * 2 > image->width){
    free(parametrs);
    puts("Too big width");
    return;
}

int y;
for(y = 0; y < i_width; y++){
    png_byte *row = image->row_pointers[y];
    for(int x = 0; x < image->width; x++){
        png_byte *ptr = &(row[indicator*x]);
        if(y % 16 <= 7 && x % 16 <= 7){
            ptr[0] = parametrs[0];
            ptr[1] = parametrs[1];
            ptr[2] = parametrs[2];
            if(indicator == 4) ptr[3] = parametrs[3];
        }
        else if(y % 16 > 7 && x % 16 > 7){
            ptr[0] = parametrs[0];
            ptr[1] = parametrs[1];
            ptr[2] = parametrs[2];
            if(indicator == 4) ptr[3] = parametrs[3];
        }
    }
}
for(y; y < image->height - i_width; y++){
    png_byte *row = image->row_pointers[y];
    for(int x = 0; x < image->width; x++){
        png_byte *ptr = &(row[indicator*x]);
        if(x < i_width || x >= image->width - i_width){
            if(y % 16 <= 7 && x % 16 <= 7){
                ptr[0] = parametrs[0];
                ptr[1] = parametrs[1];
                ptr[2] = parametrs[2];
```

```
                if(indicator == 4) ptr[3] = parametrs[3];
            }
            else if(y % 16 > 7 && x % 16 > 7){
                ptr[0] = parametrs[0];
                ptr[1] = parametrs[1];
                ptr[2] = parametrs[2];
                if(indicator == 4) ptr[3] = parametrs[3];
            }
        }
    }
    for(y; y < image->height; y++){
        png_byte *row = image->row_pointers[y];
        for(int x = 0; x < image->width; x++){
            png_byte *ptr = &(row[indicator*x]);
            if(y % 16 <= 7 && x % 16 <= 7){
                ptr[0] = parametrs[0];
                ptr[1] = parametrs[1];
                ptr[2] = parametrs[2];
                if(indicator == 4) ptr[3] = parametrs[3];
            }
            else if(y % 16 > 7 && x % 16 > 7){
                ptr[0] = parametrs[0];
                ptr[1] = parametrs[1];
                ptr[2] = parametrs[2];
                if(indicator == 4) ptr[3] = parametrs[3];
            }
        }
    }
    free(parametrs);
}

void BresenhamAlgorithm(sPng *image, int x1, int y1, int x2, int y2, int *params,
int indicator){
    const int deltaX = abs(x2 - x1);
    const int deltaY = abs(y2 - y1);
    const int signX = x1 < x2 ? 1 : -1;
    const int signY = y1 < y2 ? 1 : -1;
    int error = deltaX - deltaY;
```

```
    png_byte *ptry = image->row_pointers[y2];
    png_byte *ptrx = &(ptry[x2*indicator]);

    for(int i = 0; i < indicator; i++)
        ptrx[i] = params[i];

    while (x1 != x2 || y1 != y2){
        png_byte *row = image->row_pointers[y1];
        png_byte *ptr = &(row[x1*indicator]);

        for(int i = 0; i < indicator; i++)
            ptr[i] = params[i];
        int error2 = error * 2;
        if(error2 > -deltaY){
            error -= deltaY;
            x1 += signX;
        }
        if(error2 < deltaX){
            error += deltaX;
            y1 += signY;
        }
    }

}

void drawTunnel(sPng *image, char *color, char *width, int indicator){
    char colours[10][20] = {"red", "green", "blue", "gray", "yellow", "purple",
"white", "black", "cyan", "orange"};
    int ind = -1;
    for(int i = 0; i < 10; i++)
        if(!strcmp(color, colours[i])) ind = i;
    int *parametrs = (int *)calloc(4, sizeof(int));
    if(!setParams(parametrs, ind, color)){
        free(parametrs);
        return;
    }

    if(!isNum(width)){
        free(parametrs);
```

```
        puts("Wrong width");
        return;
    }
    int i_width = atoi(width);


    if(image->height < 2 * i_width || image->width < 2 * i_width){
        free(parametrs);
        puts("Too big width");
        return;
    }
    int small_x = image->width - 2 * i_width;
    int small_y = image->height - 2 * i_width;

    int line_width = (image->width/50) ? (image->width/50) : 2;
    int space_width = (image->width/200) ? (image->width/200) : 2;

    for(int y1 = 0; y1 < image->height; ++y1){
        if(y1 % line_width >= space_width){
            int y2 = (int)(small_y * (double)y1/image->height);
            BresenhamAlgorithm(image, 0, y1, i_width - 1, y2 + i_width, parametrs,
indicator);
            BresenhamAlgorithm(image, image->width - 1, y1, image->width -
i_width, y2 + i_width, parametrs, indicator);

        }
    }

    for(int x1 = 0; x1 < image->width; ++x1){
        if(x1 % line_width > space_width){
            int x2 = (int)(small_x * (double)x1/image->width);
            BresenhamAlgorithm(image, x1, 0, x2 + i_width, i_width - 1, parametrs,
indicator);
            BresenhamAlgorithm(image, x1, image->height - 1, x2 + i_width,
image->height - i_width, parametrs, indicator);
        }
    }
    free(parametrs);
}
```

```
void makeFramePNG(sPng *image, char *frame_type, char *width, char *color){
    int color_indicator = checkColorType(image);
    if(color_indicator == -1){
        puts("Wrong color indicator expected RGB or RGBA");
        return;
    }
    char frames[4][20] = {"common", "fractal", "tunnel", "chess"};
    int flag = -1;
    for(int i = 0; i < 4; i++){
        if(!strcmp(frame_type, frames[i])){
            flag = i;
            break;
        }
    }
    switch (flag)
    {
        case -1:
            puts("No such frame");
            break;
        case 0:
            drawCommon(image, color, width, color_indicator);
            break;
        case 1:
            drawFractal(image, color, width, color_indicator);
            break;
        case 2:
            drawTunnel(image, color, width, color_indicator);
            break;
        case 3:
            drawChess(image, color, width, color_indicator);
            break;
        default:
            break;
    }
}

void makeFrameGet(int argc, char **argv, int *opt_index, sPng *image, char
*filename){
    const char *opts = "t:c:w:?";
```

```
    struct option lOpt[] = {
        {"type", required_argument, NULL, 't'},
        {"color", required_argument, NULL, 'c'},
        {"width", required_argument, NULL, 'w'},
        {NULL, 0, NULL, 0}
    };
    char *frame_type = NULL, *width = NULL, *color = NULL;
    int opt;
    while (-1 != (opt = getopt_long(argc, argv, opts, lOpt, opt_index)))
    {
        switch(opt){
            case 't':
                frame_type = optarg;
                break;
            case 'c':
                color = optarg;
                break;
            case 'w':
                width = optarg;
                break;
            case '?':
            default:
                printHelp(4);
                writePNG(filename, image);
                exit(0);
                break;
        }
    }
    if(frame_type == NULL || color == NULL || width == NULL){
        puts("You did not choose one or more options of frames command");
        printHelp(5);
        writePNG(filename, image);
        exit(0);
    }
    makeFramePNG(image, frame_type, width, color);
    if(!writePNG(filename, image)){
        exit(0);
    }
}
```

- **Файл info.h**

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

typedef struct Png{
   int width, height;
   png_byte color_type;
   png_byte bit_depth;

   png_structp png_ptr;
   png_infop info_ptr;
   int number_of_passes;
   png_bytep *row_pointers;
}sPng;

typedef struct Rect{
   int x, y, height, width;
}Rect;

/*----------HELPING FUNCTIONS----------*/

void printHelp(int arg);
int isNum(char *str);
int setParams(int *parametrs, int ind, char *color);
int checkColorType(sPng *image);
void info(sPng *image);

/*----------HELPING FUNCTIONS----------*/

/*----------PNG FUNCTIONS----------*/

int readPNG(char *filename, sPng *image);
int writePNG(char *filename, sPng *image);
```

```
void copyAreaPNG(sPng *image, char *left_up, char *right_down, char
*destination);
void transformPNG(sPng *image, char *color1, char *color2);
void makeFramePNG(sPng *image, char *frame_type, char *width, char *color);
void rectsPNG(sPng *image, char *rect_color, char *frame_color, char *width);

/*----------PNG FUNCTIONS----------*/

/*----------GETOPT FUNCTIONS----------*/

void copyAreaGet(int argc, char **argv, int *opt_index, sPng *image, char
*filename);
void transformGet(int argc, char **argv, int *opt_index, sPng *image, char
*filename);
void makeFrameGet(int argc, char **argv, int *opt_index, sPng *image, char
*filename);
void rectsGet(int argc, char **argv, int *opt_index, sPng *image, char *filename);

/*----------GETOPT FUNCTIONS----------*/
```