**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МОЭВМ**

**КУРСОВАЯ РАБОТА**

**по дисциплине «Программирование»**

**Тема: Обработка изображения в формате PNG**

| Студент гр. 1303 |  | Чубан Д.В. |
| --- | --- | --- |
| Преподаватель |  | Чайка К.В. |

Санкт-Петербург

2022

**ЗАДАНИЕ**

**НА КУРСОВУЮ РАБОТУ**

| Студент: Чубан Дмитрий |
| --- |
| Группа 1303 |
| Тема работы: Обработка изображения в формате **PNG**  **Вариант 23**  Условие задания: |
| Программа должна реализовывать следующий функционал по обработке PNG-файла   * (1) Рисование окружности. Окружность определяется:   + **либо** координатами левого верхнего и правого нижнего угла квадрата, в который она вписана, **либо** координатами ее центра и радиусом   + толщиной линии окружности   + цветом линии окружности   + окружность может быть залитой или нет   + цветом которым залита сама окружность, если пользователем выбрана залитая окружность * (2) Фильтр rgb-компонент. Этот инструмент должен позволять для всего изображения либо установить в 0 либо установить в 255 значение заданной компоненты. Функционал определяется   + Какую компоненту требуется изменить   + В какой значение ее требуется изменить * (3) Разделяет изображение на N\*M частей. Реализация: либо провести линии заданной толщины, тем самым разделив изображение либо сохранение каждой части в отдельный файл. -- по желанию студента (можно и оба варианта). Функционал определяется:   + Количество частей по “оси” Y   + Количество частей по “оси” X   + Толщина линии   + Цвет линии   + Либо путь куда сохранить кусочки   Дата выдачи задания: 22.03.2022  Дата сдачи реферата: 01.03.2022  Дата защиты работы: 03.06.2022  Студент группы 1303 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Чубан Д.В.  Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Чайка К.В. |
| **АННОТАЦИЯ**  Работа представляет собой программу, осуществляющую обработку изображений типа PNG с использованием терминального интерфейса. В программе реализованы следующие опции для обработки изображения: рисование окружности, изменение одного из RGB-каналов изображение в заданную величину, разделение изображения линиями заданной толщины на заданное количество N\*M частей. |
|  |

**ВВЕДЕНИЕ**

Целью работы являлось: научиться обрабатывать изображения в языке программирования Си, изучить и освоить передачу опций и аргументов к ним через командную строку. Написать программу, осуществляющую считывание изображения, обработку и сохранение полученного результата в новый файл.

**1. ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ**

* 1. **Структуры**

Для работы программы были написаны структуры *Png, Configs.*

В структуру *Png* основная информация об изображении. *Configs* - структура, хранящая в себе флаги и аргументы, переданные с опциями.

* 1. **Описание функции printInfo()**

Функция выводит основную информацию об изображении (ширина, высота, битовая глубина и тип цветовой палитры).

* 1. **Описание функций read\_png\_file() и write\_png\_file()**

Данные функции обеспечивают считывание информации об изображении и ее обновлении после изменений, сделанных в ходе работы программы.

* 1. **Описание функции drawPixel()**

Функция принимает на вход ссылку на изображение Png\* image, координаты пикселя, цвет которого надо изменить (int x, int y) и 3 значения для каждого канала RGB. После этого проверяется, находится ли данный пиксель в пределах изображения, и изменяются значения RGB.

* 1. **Описание функции checkColor()**

Функция принимает на вход 3 значения типа int, соответствующие значениям цвета, и проверяет, не выходят ли данные значения за допустимые границы (0-255).

* 1. **Описание функции** **drawOutline()**

Функция принимает на вход изображение, радиус рисуемого круга, координаты его центра, три RGB цвета, определяющие цвет линии и двумерный массив int mask, который нужен для определения заливаемых пикселей (если пользователь выбрал опцию заливки и/или толщины окружности).

Далее алгоритмом Брезенхема рисуется окружность с помощью функции drawPixel() и отмечает в mask перекрашенные пиксели.

* 1. **Описание функции Fillment()**

Функция принимает на вход изображение, маску заливки, координаты начала заливки и ее цвет.

Функция проверяет, находится ли текущий пиксель в границах изображения и равенство 0 соответствующему элементу из маски, после чего меняет цвет текущего пикселя на заливаемый с помощью drawPixel() и рекурсивно вызывает себя для соседних пикселей.

* 1. **Описание функции strtokForTwo()**

Функция принимает на вход строку и два значения, куда нужно будет записать разделенные значения.

Функция делит строку с помощью функции strtok() и, если значений в строке больше или меньше двух, уведомляет о неправильности введенных данных и завершает программу.

* 1. **Описание функции strtokForThree()**

Функция принимает на вход строку и три значения, куда нужно будет записать разделенные значения.

Функция делит строку с помощью функции strtok() и, если значений в строке больше или меньше трех, уведомляет о неправильности введенных данных и завершает программу.

* 1. **Описание функции drawCircle()**

Функция принимает на вход изображение и struct Configs cfg, откуда будут браться данные для рисования окружности.

Далее, инициализируется двумерный массив int mask, который нужен для выполнения заливки и создании толщины линии окружности. Если окружность была задана пользователем с помощью координат углов квадрата, в который вписана окружность, то функция просчитывает центр этой окружности и ее радиус.

Далее вызывается функция drawOutline() и если пользователь выбрал опцию толщины, то дополнительно вызываются еще одна drawOutline() и Fillment(). Если выбрана опция заливки, то выполняется функция Fillment().

* 1. **Описание функции colorChanger()**

Функция принимает на вход изображение, букву, соответствующую одному из каналов RGB и значение, на которое данный канал нужно изменить. Если значения входных данных не соответствуют правильному формату, выводится сообщение об ошибке введенных данных и программа завершается.

Функция проверяет на валидность введенные букву и значение цвета, после чего в цикле меняет значение RGB канала для каждого пикселя на изображении.

* 1. **Описание функции divideImage()**

Функция принимает на вход изображение и структуру конфигураций.

После чего в цикле на изображении с помощью функции drawPixel() рисуются линии-разделители заданной толщины (если она была задана).

* 1. **Описание функции main()**

В главной функции происходит инициализация структуры конфигураций стандартными значениями, потом вызывается функция optsProcessing(), которая заполняет структуру данными. Далее, если в программу при запуске не было передано значение входного, выводится ошибка и программа завершается. После проверки вызывается функция read\_png\_file() и ее результат передается в переменную image. Далее в зависимости от введенных опций вызывается одна из функций printInfo(), drawCircle(), divideImage() или colorChanger().

После выполнения одной из функций, результат преобразований записывается в новый файл с помощью функции write\_png\_file() либо с именем, введенным пользователем, либо со стандартным output.png.

* 1. **Описание функции printHelp()**

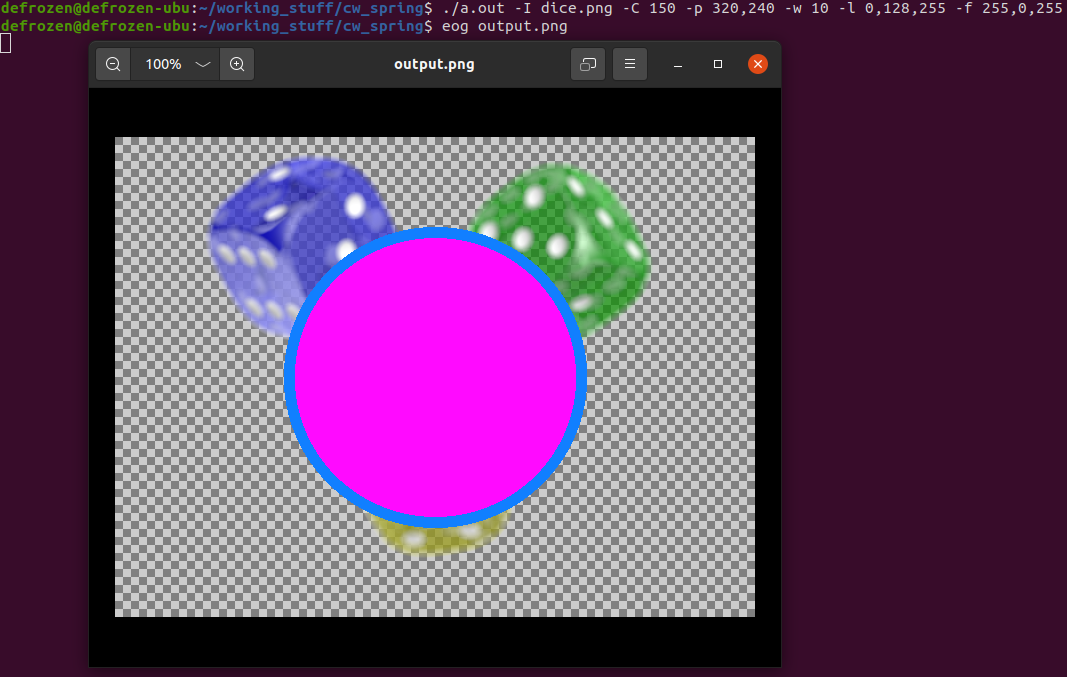
Функция выводит справку об использовании программы со всеми возможными ключами.

* 1. **Описание функции optsProcessing()**

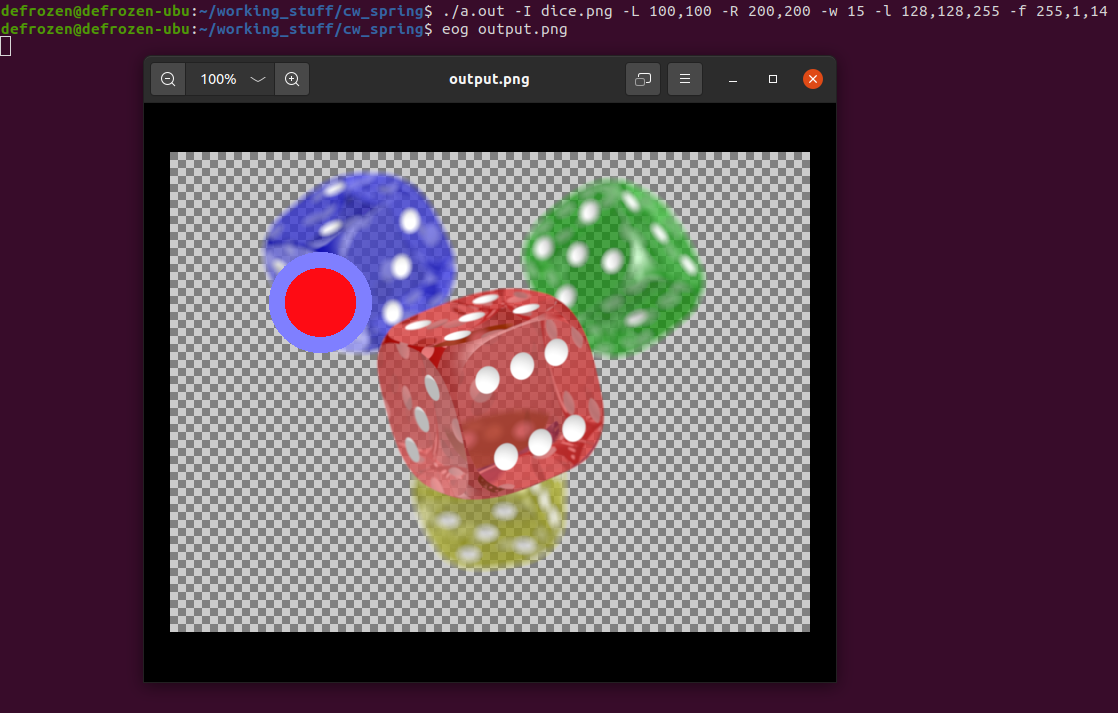
Функция разбирает введенные аргументы и записывает данные в файл конфигурации.

**2. ТЕСТИРОВАНИЕ**

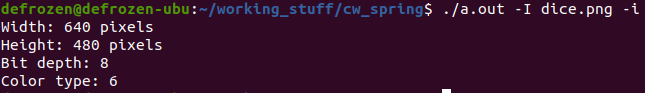
**2.1. Тестирование опции -C (--circle)**



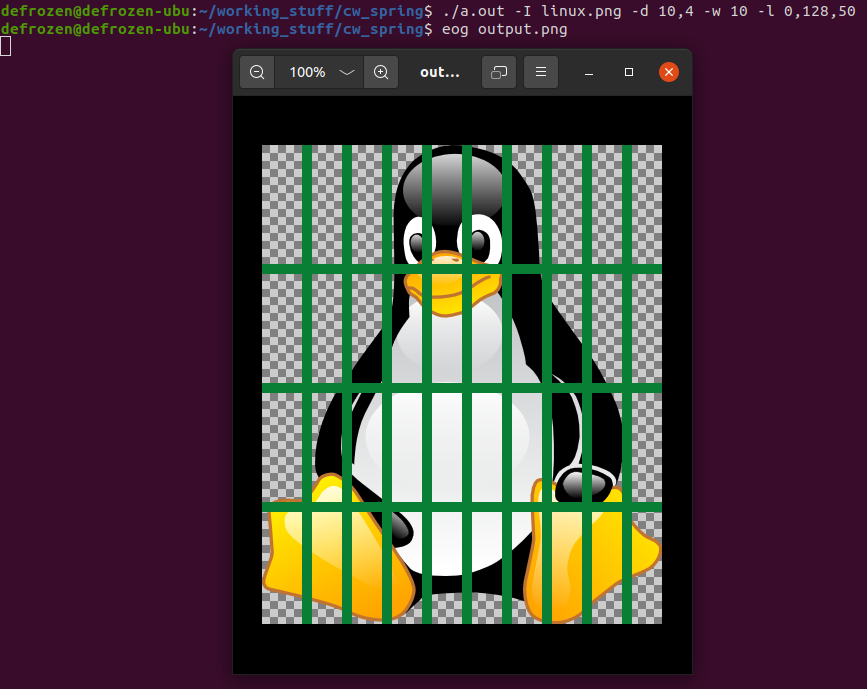
**2.2. Тестирование опции -L -R (--lt –rb)**

****

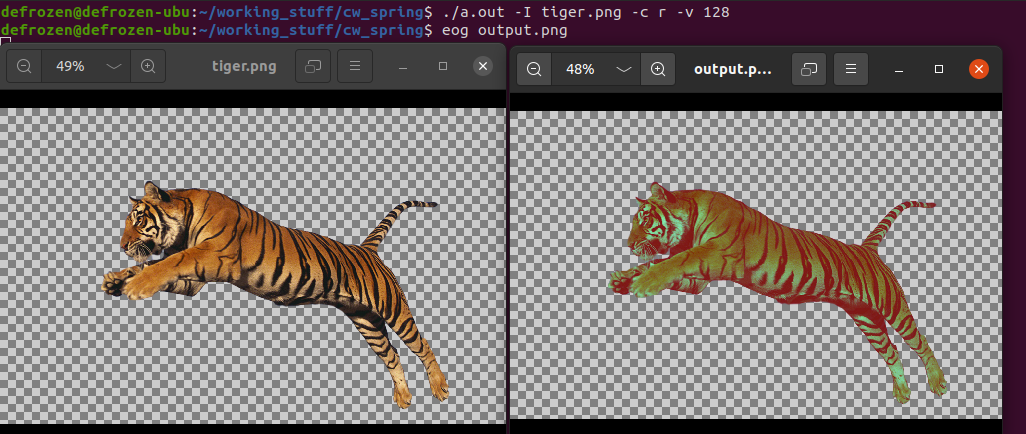
**2.3. Тестирование опции -i (--info)**

****

**2.4. Тестирование опции -d (--divide)**

****

**2.5. Тестирование опции -c (--color)**



**ЗАКЛЮЧЕНИЕ**

В ходе выполнения работы была написана программа, выполняющая считывание, обработку PNG изображения, а также сохранение полученного результата в новый файл. Программа проводит проверку корректности поданных на вход аргументов и выводит соответствующие сообщения. Взаимодействие пользователя и утилиты осуществляется через CLI (Command line interface).

**ПРИЛОЖЕНИЕ А**

Название файла: cw.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <png.h>

#include <unistd.h>

#include <getopt.h>

#include <ctype.h>

void printHelp(){

printf("-I --input - filename to open\n");

printf("-O --output - filename to save\n");

printf("-i --info - prints general information about picture \n\n");

printf("\tTo draw a circle:\n\n");

printf("-C --circle, arg - radius, amount of pixels\n");

printf("-p --pos - coordinates of center !MUST BE WITHIN PICTURE OR ELSE WON'T WORK!\n");

printf("\t\tOR\n");

printf("-L --lt in /X,Y/ format for right top corner AND -R --rb in /X,Y/ format for right bottom corner\n");

printf("WARNING: corners must be WITHIN picture AND form a square!\n\n");

printf("-w, --width\n");

printf("-l --linecolor - in format /R,G,B/\n");

printf("-f --fill - in format /R,G,B/\n\n");

printf("\tTo change the overall color of picture:\n");

printf("-c --color - in format /R/ or /G/ or /B/\n");

printf("-v --value - sets the value of color, in range 0-255\n\n");

printf("\tTo divide the picture in frames\n");

printf("-d --divide - args in format /X,Y/, should be less than height or width divided by 2\n");

printf("-w, --width\n");

printf("-l --linecolor - in format /R,G,B/\n");

};

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;

};

struct Configs{

int cfg\_error\_flag;

int info\_flag;

char\* name;

char\* output\_name;

// CIRCLE

int circle;

char\* position;

char\* left\_top\_corner;

char\* right\_bot\_corner;

int width;

char\* linecolor;

char\* fill;

//COLOR CHANGING

char color;

int var;

//DIVIDING

char\* div;

//char\* linecolor

//int width

};

void optsProcessing (int argc, char\* argv[], struct Configs \*config){

char \*opts = "hI:O:iC:p:w:l:f:c:v:d:L:R:";

struct option longOpts[]={

{"help", no\_argument, 0, 'h'},

{"input", required\_argument, 0, 'I'},

{"output", required\_argument, 0, 'O'},

{"info", no\_argument, 0, 'i'},

{"circle", required\_argument, 0, 'C'},

{"pos", required\_argument, 0, 'p'},

{"width", required\_argument, 0, 'w'},

{"linecolor", required\_argument, 0, 'l'},

{"fill", required\_argument, 0, 'f'},

{"color", required\_argument, 0, 'c'},

{"value", required\_argument, 0, 'v'},

{"divide", required\_argument, 0, 'd'},

{"lt", required\_argument, 0, 'L'},

{"rb", required\_argument, 0, 'R'},

{0, 0, 0, 0}

};

int opt;

int longIndex;

opt = getopt\_long(argc, argv, opts, longOpts, &longIndex);

while(opt != -1){

switch(opt){

case 'h':

printHelp();

exit(1);

break;

case 'I':

config->name = optarg;

break;

case 'C':

config->circle = atoi(optarg);

break;

case 'p':

config->position = optarg;

break;

case 'w':

config->width = atoi(optarg);

break;

case 'f':

config->fill = optarg;

break;

case 'l':

config->linecolor = optarg;

break;

case 'c':

if(strlen(optarg) == 1){

config->color = optarg[0];

}else{

printf("Wrong input format of color\n");

printHelp();

}

break;

case 'v':

config->var = atoi(optarg);

break;

case 'd':

config->div = optarg;

break;

case 'O':

config->output\_name = optarg;

break;

case 'L':

config->left\_top\_corner = optarg;

break;

case 'R':

config->right\_bot\_corner = optarg;

break;

case 'i':

config->info\_flag = 1;

break;

}

opt = getopt\_long(argc, argv, opts, longOpts, &longIndex);

}

argc -= optind;

argv += optind;

};

void read\_png\_file(char \*file\_name, struct Png \*image) {

int x,y;

char header[8]; // 8 is the maximum size that can be checked

/\* open file and test for it being a png \*/

FILE \*fp = fopen(file\_name, "rb");

if(!fp){

printf("No such file!\n");

exit(1);

}

fread(header, 1, 8, fp);

image->png\_ptr = png\_create\_read\_struct(PNG\_LIBPNG\_VER\_STRING, NULL, NULL, NULL);

image->info\_ptr = png\_create\_info\_struct(image->png\_ptr);

if (setjmp(png\_jmpbuf(image->png\_ptr))){

// Some error handling: error during init\_io

printf("error during init\_io\n");

exit(1);

}

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);

/\* read file \*/

if (setjmp(png\_jmpbuf(image->png\_ptr))){

// Some error handling: error during read\_image

printf("error during read\_image\n");

exit(1);

}

image->row\_pointers = (png\_bytep \*) malloc(sizeof(png\_bytep) \* image->height);

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

image->row\_pointers[y] = (png\_byte \*) malloc(png\_get\_rowbytes(image->png\_ptr, image->info\_ptr));

png\_read\_image(image->png\_ptr, image->row\_pointers);

fclose(fp);

}

void write\_png\_file(char \*file\_name, struct Png \*image) {

int x,y;

/\* create file \*/

FILE \*fp = fopen(file\_name, "wb");

/\* initialize stuff \*/

image->png\_ptr = png\_create\_write\_struct(PNG\_LIBPNG\_VER\_STRING, NULL, NULL, NULL);

image->info\_ptr = png\_create\_info\_struct(image->png\_ptr);

if (setjmp(png\_jmpbuf(image->png\_ptr))){

// Some error handling: error during init\_io

printf("error during init\_io\n");

exit(1);

}

png\_init\_io(image->png\_ptr, fp);

/\* write header \*/

if (setjmp(png\_jmpbuf(image->png\_ptr))){

// Some error handling: error during writing header

printf("error during writing header\n");

exit(1);

}

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);

/\* write bytes \*/

if (setjmp(png\_jmpbuf(image->png\_ptr))){

// Some error handling: error during writing bytes

printf("error during writing bytes\n");

exit(1);

}

png\_write\_image(image->png\_ptr, image->row\_pointers);

/\* end write \*/

if (setjmp(png\_jmpbuf(image->png\_ptr))){

// Some error handling: error during end of write

printf("error during end of write\n");

exit(1);

}

png\_write\_end(image->png\_ptr, NULL);

/\* cleanup heap allocation \*/

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

free(image->row\_pointers[y]);

free(image->row\_pointers);

fclose(fp);

}

void printInfo(struct Png\* image){

printf("Width: %d pixels\n", image->width);

printf("Height: %d pixels\n", image->height);

printf("Bit depth: %d\n", image->bit\_depth);

printf("Color type: %d\n", image->color\_type);

}

void checkColor(int R, int G, int B){

int errFlag = 0;

if(R < 0 || R > 255 ){

printf("Red color channel value is incorrect. Check the input data.\n");

errFlag = 1;

};

if(G < 0 || G > 255 ){

printf("Green color channel value is incorrect. Check the input data.\n");

errFlag = 1;

};

if(B < 0 || B > 255 ){

printf("Blue color channel value is incorrect. Check the input data.\n");

errFlag = 1;

};

if(errFlag){

exit(1);

};

}

void drawPixel(struct Png \*image, int x, int y, int color\_R, int color\_G, int color\_B){

if(x >= 0 && y >=0 && x < image->width && y < image->height){

png\_byte \*row = image->row\_pointers[y];

png\_byte \*ptr = &(row[x\*4]);

ptr[0] = color\_R;

ptr[1] = color\_G;

ptr[2] = color\_B;

ptr[3] = 255;

}

}

void drawOutline(struct Png \*image,int rad, int x, int y, int line\_R, int line\_G, int line\_B, int\*\* mask){

int currX = 0;

int currY = rad;

int delta = 1 - 2 \* rad;

int error = 0;

while(currY >= 0) {

drawPixel(image, x+currX, y+currY, line\_R, line\_G, line\_B);

mask[y+currY][x+currX] = 1;

drawPixel(image, x+currX, y-currY, line\_R, line\_G, line\_B);

mask[y+currY][x-currX] = 1;

drawPixel(image, x-currX, y+currY, line\_R, line\_G, line\_B);

mask[y-currY][x+currX] = 1;

drawPixel(image, x-currX, y-currY, line\_R, line\_G, line\_B);

mask[y-currY][x-currX] = 1;

error = 2 \* (delta + currY) - 1;

if(delta < 0 && error <= 0) {

++currX;

delta += 2 \* currX + 1;

continue;

}

error = 2 \* (delta - currX) - 1;

if(delta > 0 && error > 0) {

--currY;

delta += 1 - 2 \* currY;

continue;

}

++currX;

delta += 2 \* (currX-currY);

--currY;

}

}

void Fillment(struct Png\* image, int\*\* mask, int x, int y, int fill\_R, int fill\_G, int fill\_B){

if(x >= 0 && x < image->width && y >=0 && y < image->height){

png\_byte \*row = image->row\_pointers[y];

png\_byte \*ptr = &(row[x\*4]);

if (x >= 0 && x <= image->width && y>=0 && y <= image->height && mask[y][x] == 0){

drawPixel(image, x, y, fill\_R, fill\_G, fill\_B);

mask[y][x] = 1;

Fillment(image, mask, x+1, y, fill\_R, fill\_G, fill\_B);

Fillment(image, mask, x, y+1, fill\_R, fill\_G, fill\_B);

Fillment(image, mask, x-1, y, fill\_R, fill\_G, fill\_B);

Fillment(image, mask, x, y-1, fill\_R, fill\_G, fill\_B);

};

};

}

void strtokForTwo(char\* mergedPos, int\* x, int\* y){

char\* pch;

pch = strtok(mergedPos, ",");

if(pch != NULL){

\*x = atoi(pch);

}else{

printf("Wrong input format!\n\n");

printHelp;

exit(1);

};

pch = strtok(NULL, ",");

if(pch != NULL){

\*y = atoi(pch);

}else{

printf("Wrong input format!\n\n");

printHelp();

exit(1);

};

pch = strtok(NULL, ",");

if(pch != NULL){

printf("Wrong input format!\n\n");

printHelp();

exit(1);

}

}

void strtokForThree(char\* mergedColors, int\* R, int\* G, int\* B){

char\* pch = strtok(mergedColors, ",");

if(pch != NULL){

\*R = atoi(pch);

}else{

printf("Wrong input format!\n");

printHelp();

return;

};

pch = strtok(NULL, ",");

if(pch != NULL){

\*G = atoi(pch);

}else{

printf("Wrong input format!\n");

printHelp();

return;

};

pch = strtok(NULL, ",");

if(pch != NULL){

\*B = atoi(pch);

}else{

printf("Wrong input format!\n");

printHelp();

return;

};

pch = strtok(NULL, ",");

if(pch != NULL){

printf("Wrong input format!\n\n");

printHelp();

return;

}

}

void drawCircle(struct Png \*image, struct Configs cfg){

int\* mask[image->height];

for(int i = 0; i < image->height; i++){

mask[i] = malloc(image->width\*sizeof(int));

};

int x;

int y;

char\* pch;

if(cfg.circle != 0){

char\* mergedPos = malloc(101\*sizeof(char));

strcpy(mergedPos, cfg.position);

strtokForTwo(mergedPos, &x, &y);

free(mergedPos);

}else{

int LT\_x;

int LT\_y;

int RB\_x;

int RB\_y;

char\* mergedPosLT = malloc(101\*sizeof(char));

strcpy(mergedPosLT, cfg.left\_top\_corner);

strtokForTwo(mergedPosLT, &LT\_x, &LT\_y);

free(mergedPosLT);

char\* mergedPosRB = malloc(101\*sizeof(char));

strcpy(mergedPosRB, cfg.right\_bot\_corner);

strtokForTwo(mergedPosRB, &RB\_x, &RB\_y);

free(mergedPosRB);

if(RB\_x - LT\_x != RB\_y - LT\_y){

printf("Entered coordinates do not form a square! Check your entered coordinates!\n");

return;

}

x = LT\_x + (RB\_x - LT\_x) / 2;

y = LT\_y + (RB\_y - LT\_y) / 2;

cfg.circle = (RB\_x - LT\_x) / 2;

}

if(x <= 0 || x >= image->width || y <= 0 || y >= image->height){

printf("Center of circle is outside of picture.\n");

printHelp();

return;

};

int line\_R;

int line\_G;

int line\_B;

char\* mergedColors = malloc(12\*sizeof(char));

strcpy(mergedColors, cfg.linecolor);

strtokForThree(mergedColors, &line\_R, &line\_G, &line\_B);

free(mergedColors);

if(line\_R == 0 && line\_G == 0 && line\_B == 0){

line\_R = 1;

line\_B = 1;

line\_G = 1;

}

checkColor(line\_R, line\_G, line\_B);

int fillFlag = 0;

int fill\_R;

int fill\_G;

int fill\_B;

if (strcmp(cfg.fill, "") != 0){

fillFlag = 1;

char\* mergedFill = malloc(12\*sizeof(char));

strcpy(mergedFill, cfg.fill);

strtokForThree(mergedFill, &fill\_R, &fill\_G, &fill\_B);

free(mergedFill);

if(fill\_R == 0 && fill\_G == 0 && fill\_B == 0){

fill\_R = 1;

fill\_B = 1;

fill\_G = 1;

}

checkColor(fill\_R, fill\_G, fill\_B);

}

// DRAWING CIRCLE

drawOutline(image, cfg.circle, x, y, line\_R, line\_G, line\_B, mask);

if(cfg.width > 2){

drawOutline(image, cfg.circle-cfg.width, x, y, line\_R, line\_G, line\_B, mask);

if(x < image->width && y < image->height && x > 0 && y >0){

if(x <= (image->width / 2)){

Fillment(image, mask, x+cfg.circle-1, y, line\_R, line\_G, line\_B);

};

if(x > (image->width / 2)){

Fillment(image, mask, x-cfg.circle+1, y, line\_R, line\_G, line\_B);

};

};

};

if (fillFlag == 1){

Fillment(image, mask, x, y, fill\_R, fill\_G, fill\_B);

}

}

void colorChanger(struct Png \*image, char color, int val){

if(val < 0 || val > 255){

printf("Wrong value input format!\n");

printHelp();

}

int x,y;

color = tolower(color);

if(color != 'r' && color != 'g' && color != 'b'){

printf("No such RGB channel. Check the input data.\n");

exit(1);

}

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

png\_byte \*row = image->row\_pointers[y];

for (x = 0; x < image->width; x++){

png\_byte \*ptr = &(row[x\*4]);

if (color == 'r'){

ptr[0] = val;

};

if (color == 'g'){

ptr[1] = val;

};

if (color == 'b'){

ptr[2] = val;

};

};

};

};

void divideImage(struct Png \*image, struct Configs cfg){

int amount\_x, amount\_y;

char\* mergedAmounts = malloc(10\*sizeof(char));

strcpy(mergedAmounts, cfg.div);

strtokForTwo(mergedAmounts, &amount\_x, &amount\_y);

free(mergedAmounts);

if(amount\_x == 0){

amount\_x = 1;

};

if(amount\_y == 0){

amount\_y = 1;

}

if(amount\_x > image->width/2 || amount\_y > image->height/2){

printf("Too much parts. Choose amounts less than width and height.\n");

exit(1);

}

int line\_R;

int line\_G;

int line\_B;

char\* mergedColors = malloc(12\*sizeof(char));

strcpy(mergedColors, cfg.linecolor);

strtokForThree(mergedColors, &line\_R, &line\_G, &line\_B);

free(mergedColors);

if(line\_R == 0 && line\_G == 0 && line\_B == 0){

line\_R = 1;

line\_B = 1;

line\_G = 1;

}

checkColor(line\_R, line\_G, line\_B);

int step\_x = image->width / amount\_x;

int step\_y = image->height / amount\_y;

for (int i = 0; i < image->height; i++){

int curr\_x = 0;

for (int j = 0; j < image->width; j++){

if(j % step\_x == 0 && j != 0){

drawPixel(image, j, i, line\_R, line\_G, line\_B);

curr\_x++;

for(int k = j; k < j+cfg.width; k++){

drawPixel(image, k, i, line\_R, line\_G, line\_B);

}

if (curr\_x == amount\_x-1){

break;

}

}

}

};

int curr\_y = 0;

for( int i = 0; i < image->height; i++){

if (i % step\_y == 0 && i != 0){

for(int j = 0; j < image->width; j++){

drawPixel(image, j, i, line\_R, line\_G, line\_B);

for(int k = i; k < i+cfg.width; k++){

drawPixel(image, j, k, line\_R, line\_G, line\_B);

}

}

curr\_y++;

}

if(curr\_y == amount\_y-1){

break;

}

}

}

int main(int argc, char\* argv[]){

struct Configs cfg = {0, 0, "", "output.png", 0, "1,1", "", "", 0, "0,0,0", "", 0, 255, ""};

optsProcessing(argc, argv, &cfg);

if(!strcmp(cfg.name, "")){

printf("No file on input\n");

printHelp();

return 0;

}

if(cfg.cfg\_error\_flag == 1){

return 0;

}

struct Png image;

read\_png\_file(cfg.name,&image);

if(cfg.info\_flag == 1){

printInfo(&image);

exit(0);

};

if(cfg.circle != 0 || (strcmp(cfg.left\_top\_corner, "")) && (strcmp(cfg.right\_bot\_corner, ""))){

drawCircle(&image, cfg);

write\_png\_file(cfg.output\_name, &image);

return 0;

};

if(cfg.color != 0){

colorChanger(&image, cfg.color, cfg.var);

write\_png\_file(cfg.output\_name, &image);

return 0;

};

if(strcmp(cfg.div, "")){

divideImage(&image, cfg);

write\_png\_file(cfg.output\_name, &image);

return 0;

}

printf("You have entered less opts than required for processing!\n");

return 0;

}