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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Программирование на C

Студент гр. 1384 _____ Белокобыльский И. В.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург
2022

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Белокобыльский И. В.

Группа 1384

Тема работы: Программирование на C

Исходные данные:
Программа должна иметь CLI или GUI. Ей на вход подается bmp-файл, удовлетворяющий следующим условиям:

- 24 бита на цвет
- без сжатия
- файл всегда соответствует формату BMP

Содержание пояснительной записки:
Аннотация, Содержание отчета, Введение, Отчет, Примеры работы программы

Предполагаемый объем пояснительной записки:
Не менее 10 страниц.

Дата выдачи задания: 22.03.2022

Дата сдачи реферата: 28.05.2022

Дата защиты реферата: 28.05.2022

Студент _____ Белокобыльский И. В.

Преподаватель _____ Жангиров Т. Р.

АННОТАЦИЯ

Программа на CLI должна быть реализована подобно стандартным Linux-утилитам. Обязательно:

- Наличие справки, которая распечатывается при вызове утилиты без аргументов или стандартными ключами
- Обработка всех возможных исключительных ситуаций
- Для всех (для которых это имеет смысл) ключей должны быть как полные, так и сокращенные версии
- Утилита должна иметь функцию печати подробной информации о bmp-файле
- Для каждого инструмента должен быть соответствующий ключ и ключи для его конфигурирования

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Задача состоит в том, чтобы разработать систему обработки BMP изображений при помощи языка программирования C. В зависимости от ввода пользователя, программа должна совершить одно из 4 действий:

1. Инверсия цвета в заданной области. Функционал определяется
    a. Координатами левого верхнего угла области
    b. Координатами правого нижнего угла области
2. Преобразовать в Ч/Б изображение (любым простым способом). Функционал определяется
    a. Координатами левого верхнего угла области
    b. Координатами правого нижнего угла области
    c. Алгоритмом, если реализовано несколько алгоритмов преобразования изображения (по желанию студента)
3. Изменение размера изображения с его обрезкой или расширением фона. Функционал определяется:
    a. Действием: увеличение или уменьшение изображения
    b. Цветом фона при увеличении изображения
    c. Точкой относительно которой производится действие: центр, левый верхний, правый верхний, левый нижний, правый нижний угол
4. Рисование отрезка. Отрезок определяется:
    a. координатами начала
    b. координатами конца
    c. цветом
    d. толщиной

Инструкция по запуску приложения:

1. Скомпилировать файлы *main.c, img.c, editor.c*
2. Вызвать созданный файл без ключей или с ключами -h, -?, --help для получения справки по работе программы

3. Основываясь на справке сформировать набор ключей и их аргументов
4. Со сформированным набором вызвать программу. Порядок ключей не имеет значения. Важно: программа за один запуск способна выполнять только одно действие

1. ЧТЕНИЕ ОПЦИЙ И ВЫЗОВ НЕОБХОДИМЫХ ФУНКЦИЙ

1.1 Чтение опций

Начало работы программы в функции *main* реализовано с помощью функции *getopt_long* со следующими ключами

- -g, --grayscale: преобразовать изображение в черно-белое
- -n, --negative: инвертировать цвета
- -r, --resize: изменить размер изображения
- -l, --line: нарисовать прямую
- -i, --info: напечатать информацию о файле
- -h, -? --help: вызвать справку. Ее также можно вызвать, если не передавать программе никаких опций
- -f, --file «путь к файлу»: записать обработанное изображение в указанный файл
- -a, --algorithm «название алгоритма»: алгоритм преобразования в черно-белый формат
- -s, --start «x»,«y»: начало действия
- -e, --end «x»,«y»: окончание действия
- -c, --color «HEX»: цвет
- -t, --thickness «значение»: толщина линии
- -w, --width «значение»: новая ширина изображения
- -b, --bigness «значение»: новая высота изображения
- -p, --position «название позиции»: точка относительно которой производится изменение размера
- -u, --url «путь к файлу»: изображение для редактирования

По умолчанию программа сохраняет изображение в файл “out.bmp”, затем в цикле *while* при помощи конструкции *switch* вызывает необходимое действие, или записывает новый путь к файлу для сохранения, или загружает изображение, или заполняет параметрами структуру *Config*, в которой поля задаются как

значения опций выше, при помощи конструкции *switch* в функции *void writeConfg(Image *img, Config *cfg, char opt)*.

При нахождении ключа «-u» программа вызывает функцию из файла *img.c Image *uploadImg(char *path)*. В ней происходит выделение памяти и считывание пикселей из файла, путь к которому передан в качестве аргумента, в структуру *Image*, состоящую из структур-заголовков *BitmapFileHeader* и *BitmapInfoHeader* и двумерного массива структур *Rgb* – самих пикселей. Для удобства и читаемости кода у структуры *Image* добавлены поля *w* и *h* – ширина и высота соответственно. Также возможна ситуация, когда в изображении размер информационного заголовка меньше структуры. Для этого была написана директива *#define MIN(a, b) (a < b ? a : b)*. В случае, если размер этого заголовка больше, данные считаются и сразу же удалятся, так как эта информация не является стандартом BMP файлов с 24-битной цветностью.

1.2 Вызов необходимых функций

Далее программа вызывает функцию *void callAction(Config cfg, Image *img*), которая в зависимости от введенной опции вызывает функции:

- *void rectEditing(Config cfg, Image *img),* в которой программа ставит значения по умолчанию для выполнения операций по изменению области изображения: алгоритм преобразования в черно-белый – *luminosity*, ширина и высота составляют размеры исходного. Затем вызывается одна из функций в зависимости от переданного ключа *void makeGrayscale(Image *img, int *coords, char *algorithm)* или *void makeNegative(Image *img, int *coords)*
- *void lineDrawing(Config cfg, Image *img),* в которой программа ставит значения по умолчанию для рисования прямой: цвет – черный (000000), толщина – 1. Затем вызывается функция *void drawLine(Image *img, int *start, int *end, int thickness, char *color)*
- *void resizing(Config cfg, Image *img),* в которой программа ставит значения по умолчанию для изменения размера изображения:

позиция изменения – *“lb”* – левый нижний угол, цвет – черный (000000), ширина и высота составляют размеры исходного. Затем вызывается функция *void resizeImg(Image *img, int width, int height, char *color, char *position)*

По завершении работы программа вызывает функции *void printInfoHeader (BitmapInfoHeader header)* и *void printFileHeader(BitmapFileHeader header),* печатающие информацию о файле, если был установлен флаг -i. Затем, если были произведены изменения на изображении или введен путь к сохранению файла, программа вызывает функцию *void saveImg(Image *img, char *path),* сохраняющую изображение на диск, работающую по тому же принципу, что и *uploadImg.*

Для вывода ошибок реализованы функции *void exitFree(Image *img) и void argsErr(Image *img, char *arg, char key)*. Первая прерывает работу программы с очисткой выделенной для изображения памяти, а вторая выводит сообщение об ошибке в переданном аргументе и вызывает первую.

2. ЗАДАЧИ ПО ОБРАБОТКЕ ИЗОБРАЖЕНИЯ

2.1. Инверсия цветов в заданной области

При помощи функции *int min(int count, ...)*, принимающей количество аргументов и сами аргументы и возвращающей минимальное значение поданных целых чисел, программа в циклах *for* ставит для каждого пикселя цвет по принципу 255 минус текущее значение цвета. Функция *min* используется для предотвращения ситуации выхода за рамки изображения.

2.2. Преобразования заданной области в черно-белый

Аналогично инверсии в таких же циклах *for* для каждого цвета пикселя функция *void makeGrayscale(Image *img, int *coords, char *algorithm)* ставит одно значение в зависимости от переданного алгоритма – *char *algorithm*. «Avg» – среднее арифметическое трех цветов пикселя. «Lightness» – среднее арифметическое минимального и максимального значения цветов в пикселе. Максимальное значение определяется благодаря функции *int max(int count, ...)*, аналогичной *min*. «Luminosity» – высчитывается по правилу: значение красного цвета умножить на 0.21, зеленого – на 0.72, синего – на 0.07.

2.3. Изменение размеров изображения с его обрезкой или расширением фона

Для считывания цвета с клавиатуры используется палитра HEX. С этой целью реализована функция *Rgb getColor(char *color, Image *img)*, работающая по следующему принципу:

- если в качестве аргумента подано шестизначное шестнадцатеричное число, программа считывает его в формате RRGGBB (R – красный, G – зеленый, B – синий)
- если подано трехзначное число, программа считывает его в формате RGB, а затем каждое значение возводит в квадрат. Например для FFF соответствуют значения для каждого цвета 15 * 15 = 255.

В функции *void resizeImg(Image *img, int width, int height, char *color, char *position)* программа с помощью *getColor* определяется цвет, который используется в дальнейшем для заполнения пустого пространства вследствие расширения изображения. Далее в зависимости от точки, относительно которой производятся изменения программа работает по следующему принципу: создается переменная, которая отвечает за отступ по вертикали и по горизонтали относительно левого нижнего угла в зависимости от введенного пользователем значения. Далее программа вызывает функцию *Rgb **_cutImg(Image *img, int oldW, int oldH, Rgb color, int padding[2])* или *Rgb **_expandImg(Image *img, int oldW, int oldH, Rgb color, int padding[2])* для обрезки или расширения изображения соответственно, определяется за счет текущей ширины и введенного пользователем значения. После этого программа копирует новый массив в старый, при этом очистив лишнюю память или (и) выделив новую.

Функция *_cutImg* в новый массив двумерный пикселей в цикле *for* по новой высоте записывает значения старого изображения, если текущая полоса пикселей находится в подходящей для этого зоне, что определяется благодаря переменной отступа – *padding*. Иначе же программа заполняет пиксели переданным цветом.

Функция *_expandImg* аналогично в новый массив двумерный пикселей в цикле *for* по новой высоте записывает значения старого изображения, если текущая полоса пикселей находится в подходящей для этого зоне. Причем благодаря переменной отступа программа заполняет пиксели переданным цветом до и после самого изображения. Иначе аналогично *_cutImg* пиксели заполняются переданным цветом.

2.4. Рисование прямой

Для рисования прямой реализована функция *void drawLine(Image *img, int *start, int *end, int thickness, char *color)*, которая при помощи функции *getColor* определяет введенный цвет и по алгоритму Брезенхема вызывает функцию *void setPixels(Image *img, int x, int y, int quantity, Rgb color)*.

Функция *setPixels* при помощи цикла *for* заполняет пиксели необходимым цветом, если они лежат в окружности диаметра *quantity*.

3. ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

Инверсия цвета в заданной области:

- Файл до преобразования: simpsonsvr.bmp
- Файл после преобразования: grayscale.bmp
- Команда: *gcc main.c img.c editor.c && ./a.out -u simpsonsvr.bmp --negative -s 100,100 --end 10000,10000 -f negative.bmp*

```
gcc main.c img.c editor.c && ./a.out -u simpsonsvr.bmp --negative -s 100,100 --end 10000,10000 -f negative.bmp
Simple image editor.
Rerun with -h for help.

Uploading image: simpsonsvr.bmp...
Making negative rectangle...
Saving new image: negative.bmp...
```

Преобразование изображение в черно-белый формат в заданной области:

- Файл до преобразования: v5.bmp
- Файл после преобразования: grayscale.bmp
- Команда: *gcc main.c img.c editor.c && ./a.out -u v5.bmp --grayscale --algorithm avg -s 0,100 --end 10000,1000 -f grayscale.bmp*

```
gcc main.c img.c editor.c && ./a.out -u v5.bmp --grayscale -s 0,100 --algorithm avg --end 10000,1000 -f grayscale.bmp
Simple image editor.
Rerun with -h for help.

Uploading image: v5.bmp...
Making black and white rectangle by algorithm avg...
Saving new image: grayscale.bmp...
```

Изменение размеров изображение, увеличение по ширине и по высоте:

- Файл до преобразования: spb.bmp
- Файл после преобразования: resize_wider.bmp
- Команда: *gcc main.c img.c editor.c && ./a.out -u spb.bmp --resize -w 2000 -b 2000 -c 602341 -p c -f resize_wider.bmp*

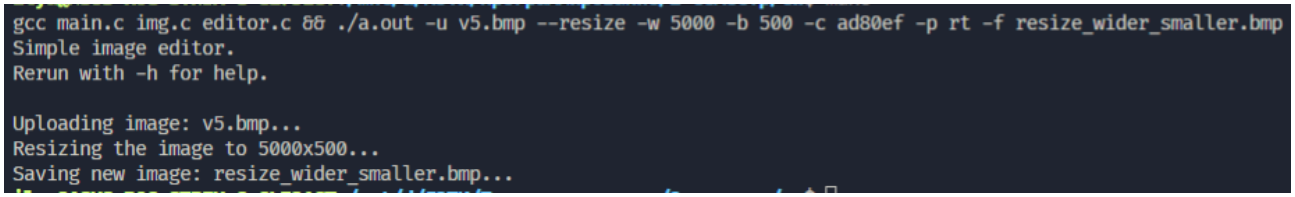
```
gcc main.c img.c editor.c && ./a.out -u spb.bmp --resize -w 2000 -b 2000 -c 602341 -p c -f resize_wider.bmp
Simple image editor.
Rerun with -h for help.

Uploading image: spb.bmp...
Resizing the image to 2000x2000...
Saving new image: resize_wider.bmp...
```

Изменение размеров изображение, увеличение по ширине и уменьшение по высоте:

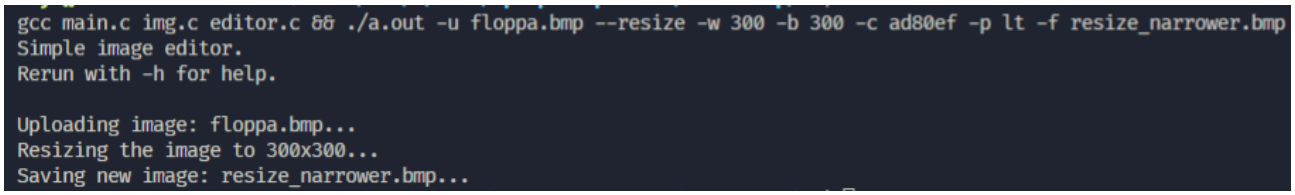
- Файл до преобразования: v5.bmp

- Файл после преобразования: resize_wider_smaller.bmp
- Команда: *gcc main.c img.c editor.c && ./a.out -u v5.bmp --resize -w 5000 -b 500 -c ad80ef -p rt -f resize_wider_smaller.bmp*



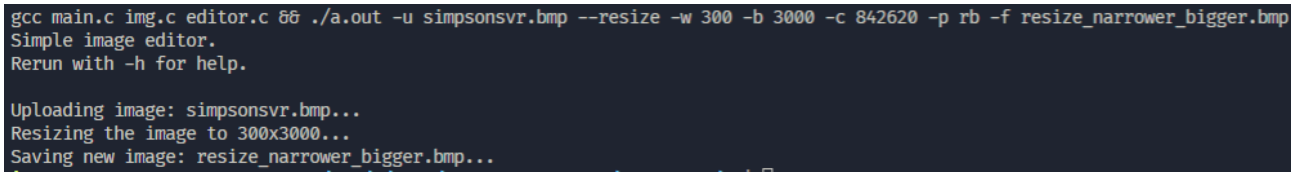
Изменение размеров изображение, уменьшение по ширине и высоте:

- Файл до преобразования: floppa.bmp
- Файл после преобразования: resize_ narrower.bmp
- Команда: *gcc main.c img.c editor.c && ./a.out -u floppa.bmp --resize -w 300 -b 300 -p lt -f resize_narrower.bmp*



Изменение размеров изображение, уменьшение по ширине и увеличение по высоте:

- Файл до преобразования: simpsonsvr.bmp
- Файл после преобразования: resize_ narrower_bigger.bmp
- Команда: *gcc main.c img.c editor.c && ./a.out -u simpsonsvr.bmp --resize -w 300 -b 3000 -c 842620 -p rb -f resize_narrower_bigger.bmp*



Рисование прямой:

- Файл до преобразования: spb.bmp
- Файл после преобразования: line.bmp
- Команда: *gcc main.c img.c editor.c && ./a.out -u spb.bmp -l -s 100,100 -e 5000,5000 -c efc9Ad -t 50 -f line.bmp*

```
gcc main.c img.c editor.c && ./a.out -u spb.bmp -l -s 100,100 -e 5000,5000 -c efc9Ad -t 50 -f line.bmp
Simple image editor.
Rerun with -h for help.

Uploading image: spb.bmp...
Drawing the line...
Saving new image: line.bmp...
```

Печать информации о файле:

- Файл: spb.bmp
- Команда: *gcc main.c img.c editor.c && ./a.out -u spb.bmp -l -s 100,100 -e 5000,5000 -c efc9Ad -t 50 -f line.bmp*

```
gcc main.c img.c editor.c && ./a.out -u spb.bmp -i
Simple image editor.
Rerun with -h for help.

Uploading image: spb.bmp...

BitmapFileHeader:
        signature:      4d42 (19778)
        filesize:       5eec8a (6220938)
        reserved1:      0 (0)
        reserved2:      0 (0)
        pixelArrOffset: 8a (138)

BitmapInfoHeader:
        headerSize:     7c (124)
        width:          780 (1920)
        height:         438 (1080)
        planes:         1 (1)
        bitsPerPixel:   18 (24)
        compression:    0 (0)
        imageSize:      5eec00 (6220800)
        xPixelsPerMeter:        ec3 (3779)
        yPixelsPerMeter:        ec3 (3779)
        colorsInColorTable:     0 (0)
        importantColorCount:    0 (0)
```

Печать справки:

- Команда: *gcc main.c img.c editor.c && ./a.out*

```
gcc main.c img.c editor.c && ./a.out
Simple image editor.
Rerun with -h for help.

Required key:
        -u --url "path/to/file.bmp" – path to image you want to edit
Optional keys without arguments:
        -h -? --help    – help
        -i --info       – information about this file
        -g --grayscale – make image in black and white. Parameters: -s --start, -e --end, -a --algorithm
        -n --negative  – make image in negative. Parameters: -s --start, -e --end
        -r --resize     – resize image. Parameters: -w --width, -b --bigness, -c --color, -p --position (default 000000)
        -l --line       – draw a line. Parameters: -s --start, -e --end, -t --thickness, -c --color (default 000000)
Optional keys with arguments:
        -f --file       "where/to/save.bmp"         – path to image where you want to be saved
        -s --start      <x coord>,<y coord>          – start of the action. Write to integer numbers separated by comma. Default 0,0
        -e --end        <x coord>,<y coord>          – end of the action. Write to integer numbers separated by comma. Default <image width>,<image height>
        -a --algorithm <luminosity|avg|lightness> – algorithm of grayscale. Default luminosity. Avg - average
        -c --color      <color in HEX>              – color of the action
        -w --width      <integer value>             – width of new image
        -b --bigness    <integer value>             – height of new image
        -p --position   <lb|rb|c|lt|rt>             – position of resizing. Left bottom | right bottom | center | left top | right top
        -t --thickness <integer value>              – thickness of the line
```

Таблица: Тестирование программы на некорректные запросы

Запрос	Ошибка	Действительный результат
gcc main.c img.c editor.c && ./a.out -u v5.bmp --grayscale -s -100,100 --end 10000,10000 -f grayscale.bmp	Отрицательное значение ключа -s	Arguments Error - incorrect type of argument: Please write arguments of -s in current format! Print -h or -? for help.
gcc main.c img.c editor.c && ./a.out -u spb.bmp -l -s 100,100 -e 5000,5000 -c efc9fAd -t 50 -f line.bmp	Некорректно введен цвет	Arguments Error - incorrect color: write color in HEX like FFFFFF: Please write arguments of -c in current format! Print -h or -? for help.
gcc main.c img.c editor.c && ./a.out -u spb.bmp -l -s 100,100 -e 5000,5000 -c zzzzzz -t 50 -f line.bmp	Некорректно введен цвет	Arguments Error - write color in HEX format: Please write arguments of -r in current format! Print -h or -? for help.
gcc main.c img.c editor.c && ./a.out -u unknown	Изображение – не bmp файл	File Error - unknown: -u argument has to be a .bmp file! Print -h or -? for help.
gcc main.c img.c editor.c && ./a.out -u unknown.bmp	Файла не существует	This file does not exist.

gcc main.c img.c editor.c && ./a.out -u simpsonsvr.bmp -n -g	Использование больше одного действия	Sorry, this app cannot do more than one action. Please, run again.
gcc main.c img.c editor.c && ./a.out -n	Не введено изображение	You should specify an image.
gcc main.c img.c editor.c && ./a.out -u floppa.bmp -l	Не введены обязательные параметры	Arguments Error - too few arguments: Please write arguments of -l in current format! Print -h or -? for help.

ЗАКЛЮЧЕНИЕ

В ходе проделанной работы была написана программа на CLI, реализующая функции графического редактора. Для удобства хранения данных о картинке, была реализована функция *Image*, хранящая в себе двумерный массив структур *Rgb* – пикселей и структуры заголовков *BitmapInfoHeader* и *BitmapFileHeader*. Для конфигурирования действий пользователя была написана структура *Config*, значения в которую записываются при помощи функци *getopt_long* и конструкций *switch*. Для выполнения задачи были реализованы следующие функции:

- В файле img.c:
  - *void printInfoHeader(BitmapInfoHeader header);*
  - *void printFileHeader(BitmapFileHeader header);*
  - *void exitFree(Image *img);*
  - *Image *uploadImg(char *path);*
  - *void saveImg(Image *img, char *path);*
- В файле main.c:
  - *void printHelp();*
  - *void argsErr(Image *img, char *arg, char key);*
  - *void writeConfg(Image *img, Config *cfg, char opt);*
  - *void callAction(Config cfg, Image *img);*
  - *int main(int argc, char **argv);*
  - *void rectEditing(Config cfg, Image *img);*
  - *void lineDrawing(Config cfg, Image *img);*
  - *void resizing(Config cfg, Image *img);*
- В файле editor.c:
  - *void makeNegative(Image *img, int *coords);*
  - *void makeGrayscale(Image *img, int *coords, char *algorithm);*
  - *int min(int count, ...);*
  - *int max(int count, ...);*

- *void drawLine(Image *img, int *start, int *end, int thickness, char *color);*
- *void resizeImg(Image *img, int width, int height, char *color, char *position);*
- *void setPixels(Image *img, int x, int y, int quantity, Rgb color);*
- *Rgb getColor(char *color, Image *img);*
- *Rgb **_cutImg(Image *img, int oldW, int oldH, Rgb color, int padding[2]);*
- *Rgb **_expandImg(Image *img, int oldW, int oldH, Rgb color, int padding[2]);*
- *unsigned char _inRound(int x0, int y0, int r, int x1, int y1);*

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.h

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <getopt.h>
#include "img.h"
#include "editor.h"

typedef struct {
	char opt;
	int start[2];
	int end[2];
	char color[7];
	int thickness;
	int width;
	int height;
	char position[3];
	char algorithm[11];
	unsigned char hasOpts;
} Config;


void printHelp();
void argsErr(Image *img, char *arg, char key);
void writeConfg(Image *img, Config *cfg, char opt);
void callAction(Config cfg, Image *img);
int main(int argc, char **argv);
void rectEditing(Config cfg, Image *img);
void lineDrawing(Config cfg, Image *img);
void resizing(Config cfg, Image *img);
```

Название файла: main.c

```
#include "main.h"


int main(int argc, char **argv) {
      printf("Simple image editor.\nRerun with -h for help.\n\n");
      Image *img = NULL;

      char *opts = "gnrlih?f:a:s:e:c:t:w:b:p:u:";
      struct option longOpts[] = {
            {"grayscale", no_argument, NULL, 'g'},
            {"negative", no_argument, NULL, 'n'},
            {"resize", no_argument, NULL, 'r'},
            {"line", no_argument, NULL, 'l'},
            {"info", no_argument, NULL, 'i'},
            {"help", no_argument, NULL, 'h'},
            {"file", required_argument, NULL, 'f'},
            {"algorithm", required_argument, NULL, 'a'},
            {"start", required_argument, NULL, 's'},
            {"end", required_argument, NULL, 'e'},
            {"color", required_argument, NULL, 'c'},
            {"thickness", required_argument, NULL, 't'},
            {"width", required_argument, NULL, 'w'},
            {"bigness", required_argument, NULL, 'b'},
            {"position", required_argument, NULL, 'p'},
            {"url", required_argument, NULL, 'u'},
            {NULL, no_argument, NULL, 0}
      };
      int opt, longIndex;
      opt = getopt_long(argc, argv, opts, longOpts, &longIndex);
      char *savePath = calloc(8, 1);
      Config cfg = {0};
      cfg.hasOpts = 0;
      unsigned char hasChanged = 0;
      unsigned char writeInfo = 0;
      strcpy(savePath, "out.bmp");
      if (opt == -1) {
            printHelp();
      }
      while (opt != -1){
            if (!hasChanged && strchr("gnrlf", opt)) {
                  hasChanged = 1;
            }
            switch (opt) {
                  case 'i':
                        writeInfo = 1;
                        break;
                  case 'h':
                  case '?':
                        printHelp();
                        break;
                  case 'g':
                  case 'n':
                  case 'r':
                  case 'l':
                        if (cfg.opt && strchr("gnrl", cfg.opt)) {
```

```
                                    puts("Sorry, this app cannot do more
than one action. Please, run again.");
                                    exitFree(img);
                               }
                               cfg.opt = opt;
                               break;
                          case 'f':
                               savePath = realloc(savePath, strlen(optarg) +
1);
                               strcpy(savePath, optarg);
                               break;
                          case 'u':
                               if (!strstr(optarg, ".bmp") && !strstr(optarg,
".dib") && !strstr(optarg, ".rle")) {
                                    fprintf(stderr, "File Error - %s: -u
argument has to be a .bmp file! Print -h or -? for help.\n", optarg);
                                    exitFree(img);
                               }
                               img = uploadImg(optarg);
                               break;
                          default:
                               writeConfg(img, &cfg, opt);
                               break;
                     }
                     opt = getopt_long(argc, argv, opts, longOpts,
&longIndex);
               }
               if (!img && (cfg.opt || writeInfo)) {
                     puts("You should specify an image.");
                     exitFree(img);
               }
               callAction(cfg, img);
               if (writeInfo) {
                     printFileHeader(img->fileHeader);
                     printInfoHeader(img->infoHeader);
               }
               if (hasChanged) {
                     saveImg(img, savePath);
               }

               free(savePath);
               return 0;
          }


          void printHelp() {
               // Вместо пробелов можно было бы использовать \t, но так лучше
читается
               puts("Required key:");
               puts("\t-u --url \"path/to/file.bmp\" — path to image you want
to edit");
               puts("Optional keys without arguments:");
               puts("\t-h -? --help   — help");
               puts("\t-i --info      — information about this file");
               puts("\t-g --grayscale — make image in black and white.
Parameters: -s --start, -e --end, -a --algorithm");
```

```
          puts("\t-n --negative  — make image in negative. Parameters: -
s --start, -e --end");
          puts("\t-r --resize    — resize image. Parameters: -w --width,
-b --bigness, -c --color, -p --position (default 000000)");
          puts("\t-l --line      — draw a line. Parameters: -s --start,
-e --end, -t --thickness, -c --color (default 000000)");
          puts("Optional keys with arguments:");
          puts("\t-f --file      \"where/to/save.bmp\"        — path to
image where you want to be saved");
          puts("\t-s --start     <x coord>,<y coord>        — start of
the action. Write to integer numbers separated by comma. Default 0,0");
          puts("\t-e --end       <x coord>,<y coord>        — end of the
action. Write to integer numbers separated by comma. Default <image
width>,<image height>");
          puts("\t-a --algorithm <luminosity|avg|lightness> — algorithm
of grayscale. Default luminosity. Avg - average");
          puts("\t-c --color     <color in HEX>             — color of
the action");
          puts("\t-w --width     <integer value>            — width of
new image");
          puts("\t-b --bigness   <integer value>            — height of
new image");
          puts("\t-p --position  <lb|rb|c|lt|rt>            — position
of resizing. Left bottom | right bottom | center | left top | right
top");
          puts("\t-t --thickness <integer value>            — thickness
of the line");
     }


     void argsErr(Image *img, char *arg, char key) {
          fprintf(stderr, "Arguments Error - %s: Please write arguments
of -%c in current format! Print -h or -? for help.\n", arg, key);
          exitFree(img);
     }


     void writeConfg(Image *img, Config *cfg, char opt) {
          cfg->hasOpts = 1;
          char *digitOpts = "setwb";
          if (strchr(digitOpts, opt) && !isdigit(optarg[0])) {
               argsErr(img, "incorrect type of argument", opt);
          }
          char *secondPart;
          switch(opt) {
               case 's':
                    if (!strchr(optarg, ',')) {
                         argsErr(img, "incorrect type of argument",
opt);
                    }
                    secondPart = strchr(optarg, ',') + 1;
                    // isdigit(*secondPart) вызывает ошибку
                    if ((int)secondPart[0] > 47 && (int)secondPart[0] <
58) {
                         cfg->start[0] = atoi(optarg);
                         cfg->start[1] = atoi(secondPart);
                    } else {
```

```
                              argsErr(img, "incorrect type of argument",
opt);
                        }
                        break;
                  case 'e':
                        if (!strchr(optarg, ',')) {
                              argsErr(img, "incorrect type of argument",
opt);
                        }
                        secondPart = strchr(optarg, ',') + 1;
                        if ((int)secondPart[0] > 47 && (int)secondPart[0] <
58) {
                              cfg->end[0] = atoi(optarg);
                              cfg->end[1] = atoi(secondPart);
                        } else {
                              argsErr(img, "incorrect type of argument",
opt);
                        }
                        break;
                  case 'c':
                        if (strlen(optarg) != 6 && strlen(optarg) != 3) {
                              argsErr(img, "incorrect color: write color in
HEX like FFFFFF", opt);
                        }
                        strcpy(cfg->color, optarg);
                        break;
                  case 't':
                        cfg->thickness = atoi(optarg);
                        break;
                  case 'w':
                        cfg->width = atoi(optarg);
                        break;
                  case 'b':
                        cfg->height = atoi(optarg);
                        break;
                  case 'p':
                        if (strlen(optarg) > 2) {
                              argsErr(img, "incorrect position", opt);
                        }
                        strcpy(cfg->position, optarg);
                        break;
                  case 'a':
                        if (strcmp(optarg, "luminosity") && strcmp(optarg,
"avg") && strcmp(optarg, "lightness")) {
                              argsErr(img, "incorrect algorithm", opt);
                        }
                        strncpy(cfg->algorithm, optarg, 10);
                        break;
            }
      }


      void callAction(Config cfg, Image *img) {
            switch (cfg.opt) {
                  case 'g':
                  case 'n':
                        rectEditing(cfg, img);
```

```
                    break;
               case 'l':
                    lineDrawing(cfg, img);
                    break;
               case 'r':
                    resizing(cfg, img);
                    break;
          }
     }


     void resizing(Config cfg, Image *img) {
          if (!cfg.hasOpts) {
               argsErr(img, "too few arguments", 'r');
          }
          cfg.width = cfg.width ? cfg.width : img->w;
          cfg.height = cfg.height ? cfg.height : img->h;

          for (int i = 0; i < strlen(cfg.color); i++) {
               if (!strchr("1234567890abcdefABCDEF", cfg.color[i])) {
                    argsErr(img, "write color in HEX format", 'r');
               }
          }
          if (!strlen(cfg.color)) {
               strcpy(cfg.color, "000000");
          }
          if (!strlen(cfg.position)) {
               strcpy(cfg.position, "lb");
          }
          resizeImg(img, cfg.width, cfg.height, cfg.color,
cfg.position);
     }


     void lineDrawing(Config cfg, Image *img) {
          if (!cfg.hasOpts) {
               argsErr(img, "too few arguments", 'l');
          }
          for (int i = 0; i < strlen(cfg.color); i++) {
               if (!strchr("1234567890abcdefABCDEF", cfg.color[i])) {
                    argsErr(img, "write color in HEX format", 'r');
               }
          }
          if (!strlen(cfg.color)) {
               strcpy(cfg.color, "000000");
          }
          cfg.thickness = cfg.thickness ? cfg.thickness : 1;
          drawLine(img, cfg.start, cfg.end, cfg.thickness, cfg.color);
     }


     void rectEditing(Config cfg, Image *img) {
          int coords[4] = {0, 0, img->w, img->h};
          char algorithm[11] = "luminosity";
          if (cfg.hasOpts) {
               if (strlen(cfg.algorithm)) {
                    strncpy(algorithm, cfg.algorithm, 10);
```

```
            }
            coords[0] = cfg.start[0];
            coords[1] = cfg.start[1];
            coords[2] = cfg.end[0] ? cfg.end[0] : coords[2];
            coords[3] = cfg.end[1] ? cfg.end[1] : coords[3];
        }
        if (cfg.opt == 'g') {
            makeGrayscale(img, coords, algorithm);
        } else if (cfg.opt == 'n') {
            makeNegative(img, coords);
        }
}
```

Название файла: img.h

```
#include <stdio.h>
#include <stdlib.h>
#define MIN(a, b) (a < b ? a : b)

#pragma once
#pragma pack (push, 1)
typedef struct
{
      unsigned short signature;
      unsigned int filesize;
      unsigned short reserved1;
      unsigned short reserved2;
      unsigned int pixelArrOffset;
} BitmapFileHeader;

typedef struct {
    unsigned int cieXYZX;
    unsigned int ciexyzY;
    unsigned int ciexyzZ;
} CIEXYZ;

typedef struct{
    CIEXYZ ciexyzRed;
    CIEXYZ ciexyzGreen;
    CIEXYZ ciexyzBlue;
} CIEXYZTRIPLE;

typedef struct
{
      unsigned int headerSize;
      unsigned int width;
      unsigned int height;
      unsigned short planes;
      unsigned short bitsPerPixel;
      unsigned int compression;
      unsigned int imageSize;
      unsigned int xPixelsPerMeter;
      unsigned int yPixelsPerMeter;
      unsigned int colorsInColorTable;
      unsigned int importantColorCount;
      unsigned int redChannelBitmask;
      unsigned int greenChannelBitmask;
      unsigned int blueChannelBitmask;
      unsigned int alphaChannelBitmask;
      unsigned int colorSpaceType;
      CIEXYZTRIPLE colorSpaceEndpoints;
      unsigned int gammaForRedChannel;
      unsigned int gammaForGreenChannel;
      unsigned int gammaForBlueChannel;
      unsigned int intent;
      unsigned int ICCProfileData;
      unsigned int ICCProfileSize;
      unsigned int reserved;
} BitmapInfoHeader;

typedef struct
```

```
{
      unsigned char b;
      unsigned char g;
      unsigned char r;
} Rgb;


typedef struct
{
      BitmapFileHeader fileHeader;
      BitmapInfoHeader infoHeader;
      int h;
      int w;
      Rgb **pixels;
} Image;

#pragma pack(pop)


void printInfoHeader(BitmapInfoHeader header);
void printFileHeader(BitmapFileHeader header);
void exitFree(Image *img);
Image *uploadImg(char *path);
void saveImg(Image *img, char *path);
```

Название файла: img.c

```
#include "img.h"

void printFileHeader(BitmapFileHeader header){
	printf("\nBitmapFileHeader:\n");
	printf("\tsignature:\t%x (%hu)\n", header.signature,
header.signature);
	printf("\tfilesize:\t%x (%u)\n", header.filesize,
header.filesize);
	printf("\treserved1:\t%x (%hu)\n", header.reserved1,
header.reserved1);
	printf("\treserved2:\t%x (%hu)\n", header.reserved2,
header.reserved2);
	printf("\tpixelArrOffset:\t%x (%u)\n", header.pixelArrOffset,
header.pixelArrOffset);
}

void printInfoHeader(BitmapInfoHeader header){
	printf("\nBitmapInfoHeader:\n");
	printf("\theaderSize:\t%x (%u)\n", header.headerSize,
header.headerSize);
	printf("\twidth:	\t%x (%u)\n", header.width,
header.width);
	printf("\theight:	\t%x (%u)\n", header.height,
header.height);
	printf("\tplanes:	\t%x (%hu)\n", header.planes,
header.planes);
	printf("\tbitsPerPixel:\t%x (%hu)\n", header.bitsPerPixel,
header.bitsPerPixel);
	printf("\tcompression:\t%x (%u)\n", header.compression,
header.compression);
	printf("\timageSize:\t%x (%u)\n", header.imageSize,
header.imageSize);
	printf("\txPixelsPerMeter:\t%x (%u)\n",
header.xPixelsPerMeter, header.xPixelsPerMeter);
	printf("\tyPixelsPerMeter:\t%x (%u)\n",
header.yPixelsPerMeter, header.yPixelsPerMeter);
	printf("\tcolorsInColorTable:\t%x (%u)\n",
header.colorsInColorTable, header.colorsInColorTable);
	printf("\timportantColorCount:\t%x (%u)\n",
header.importantColorCount, header.importantColorCount);
}


void exitFree(Image *img) {
	if (img) {
		for(int i = 0; i < img->h; i++){
			free(img->pixels[i]);
		}
		free(img->pixels);
		free(img);
	}
	exit(EXIT_FAILURE);
}


Image *uploadImg(char *path) {
```

```
        printf("Uploading image: %s...\n", path);
        FILE *fileRead = fopen(path, "rb");
        if (fileRead == NULL) {
            fprintf(stderr, "This file does not exist.\n");
            exit(EXIT_FAILURE);
        }
        Image *img = calloc(1, sizeof(Image));
        fread(&img->fileHeader, 1, sizeof(BitmapFileHeader),
fileRead);

        unsigned int headerSize;
        fread(&headerSize, 1, sizeof(unsigned int), fileRead); //
размер заголовка

        img->infoHeader.headerSize = headerSize;

        unsigned int readSize = MIN(img->infoHeader.headerSize,
sizeof(BitmapInfoHeader));
        fread(&img->infoHeader.width, 1, readSize - sizeof(unsigned
int), fileRead);

        if (img->infoHeader.bitsPerPixel != 24 || img-
>infoHeader.colorsInColorTable) {
            fprintf(stderr, "Sorry, we do not support this color
format in bmp files.\n");
            free(img);
            exit(EXIT_FAILURE);
        }
        int trashSize = img->infoHeader.headerSize - readSize;
        if (trashSize > 0) {
            char *trash = calloc(trashSize, 1);
            fread(trash, 1, trashSize, fileRead);
            free(trash);
        }

        img->h = img->infoHeader.height;
        img->w = img->infoHeader.width;

        img->pixels = malloc(img->h * sizeof(Rgb*));
        for(int i = 0; i < img->h; i++){
            img->pixels[i] = malloc(img->w * sizeof(Rgb) + (4 -
((img->w) * 3) % 4) % 4);
            fread(img->pixels[i], 1, img->w * sizeof(Rgb) + (4 -
((img->w) * 3) % 4) % 4, fileRead);
        }
        fclose(fileRead);
        return img;
    }


    void saveImg(Image *img, char *path) {
        printf("Saving new image: %s...\n", path);
        FILE *fileSave = fopen(path, "wb");
        if (!fileSave) {
            fprintf(stderr, "Error in saving file %s.\n", path);
            exitFree(img);
            exit(EXIT_FAILURE);
```

```
            }
            img->infoHeader.height = img->h;
            img->infoHeader.width = img->w;
            fwrite(&img->fileHeader, 1, sizeof(BitmapFileHeader),
fileSave);

            unsigned int readSize = MIN(img->infoHeader.headerSize,
sizeof(BitmapInfoHeader));
            fwrite(&img->infoHeader, 1, readSize, fileSave);

            int trashSize = img->infoHeader.headerSize - readSize;
            if (trashSize > 0) {
                  char *trash = calloc(trashSize, 1);
                  fwrite(&trash, 1, trashSize, fileSave);
                  free(trash);
            }
            unsigned int w = (img->w) * sizeof(Rgb) + (4 - ((img->w) * 3)
% 4) % 4;
            for(int i = 0; i < img->h; i++){
                  fwrite(img->pixels[i], 1, w, fileSave);
                  free(img->pixels[i]);
            }
            free(img->pixels);
            free(img);
            fclose(fileSave);
      }
```

Название файла: editor.h

```
#include <stdarg.h>
#include <string.h>
#include <math.h>
#include <limits.h>
#include "img.h"
#pragma once


void makeNegative(Image *img, int *coords);
void makeGrayscale(Image *img, int *coords, char *algorithm);
int min(int count, ...);
int max(int count, ...);
void drawLine(Image *img, int *start, int *end, int thickness, char
*color);
void resizeImg(Image *img, int width, int height, char *color, char
*position);
void setPixels(Image *img, int x, int y, int quantity, Rgb color);
Rgb getColor(char *color, Image *img);
```

Название файла: editor.c

```
#include "editor.h"


void makeNegative(Image *img, int *coords) {
    puts("Making negative rectangle...");
    for (int i = img->h - min(2, coords[1], img->h) - 1; i >= img->h
- min(2, coords[3], img->h) && i >= 0; i--) {
        for (int j = min(2, coords[0], img->w); j < min(2, coords[2],
img->w); j++) {
            img->pixels[i][j].r = 255 - img->pixels[i][j].r;
            img->pixels[i][j].g = 255 - img->pixels[i][j].g;
            img->pixels[i][j].b = 255 - img->pixels[i][j].b;
        }
    }
}


void makeGrayscale(Image *img, int *coords, char *algorithm) {
    printf("Making black and white rectangle by algorithm %s...\n",
algorithm);
    for (int i = img->h - min(2, coords[1], img->h) - 1; i >= img->h
- min(2, coords[3], img->h) && i >= 0; i--) {
        for (int j = min(2, coords[0], img->w); j < min(2, coords[2],
img->w); j++) {
            int value;
            int r = img->pixels[i][j].r;
            int g = img->pixels[i][j].g;
            int b = img->pixels[i][j].b;
            if (!strcmp(algorithm, "avg")) {
                value = (r + g + b) / 3;
            } else if (!strcmp(algorithm, "lightness")) {
                value = max(3, r, g, b);
                value += min(3, r, g, b);
                value /= 2;
            } else if (!strcmp(algorithm, "luminosity")) {
                value = r * 0.21 + g * 0.72 + b * 0.07;
            } else {
```

```
                    fprintf(stderr,    "Incorrect    algorithm    %s.\n",
algorithm);
                    exitFree(img);
                }
                img->pixels[i][j].r = value;
                img->pixels[i][j].g = value;
                img->pixels[i][j].b = value;
            }
        }
    }


    Rgb **_expandImg(Image *img, int oldW, int oldH, Rgb color, int
padding[2]) {
        Rgb **newPixs = malloc(img->h * sizeof(Rgb *));
        for (int i = 0; i < img->h; i++) {
            newPixs[i] = malloc(img->w * sizeof(Rgb) + (img->w * 3) % 4);
            // До отступа заполняем нужным цветом
            for (int j = 0; j < padding[0]; j++) {
                newPixs[i][j] = color;
            }
            int start = 0;
            // Копируем, если находимся в зоне картинки
            if (i - padding[1] < oldH && i >= padding[1]) {
                for (int j = padding[0]; j < oldW + padding[0]; j++) {
                    newPixs[i][j]  =  img->pixels[i  -  padding[1]][j  -
padding[0]];
                }
                start = oldW + padding[0];
            }
            // После картинки заполняем нужным цветом
            for (int j = start; j < img->w; j++) {
                newPixs[i][j] = color;
            }
        }

        return newPixs;
    }
```

```
    Rgb **_cutImg(Image *img, int oldW, int oldH, Rgb color, int
padding[2]) {
        Rgb **newPixs = malloc(img->h * sizeof(Rgb *));
        for (int i = 0; i < img->h; i++) {
            newPixs[i] = malloc(img->w * sizeof(Rgb) + (img->w * 3) % 4);
            int start = 0;
            // Если находимся в зоне картинки, копируем нужную часть
            // Иначе заполняем цветом
            if (i - padding[1] < oldH && i >= padding[1]) {
                for (int j = padding[0]; j < img->w + padding[0]; j++) {
                    newPixs[i][j  -  padding[0]]  =  img->pixels[i  -
padding[1]][j];
                }
            } else {
                for (int j = 0; j < img->w; j++) {
                    newPixs[i][j] = color;
                }
            }
        }
        return newPixs;
    }


    void resizeImg(Image *img, int width, int height, char *colorStr,
char *position) {
        printf("Resizing the image to %dx%d...\n", width, height);
        int oldH = img->h;
        int oldW = img->w;

        img->w = width;
        img->h = height;
        Rgb color = getColor(colorStr, img);

        // Отступ относительно текущего изображения
        int padding[2] = {0};
        if (!strcmp(position, "rb")) {
```

```
            padding[0] = abs(img->w - oldW);
        } else if (!strcmp(position, "lt")) {
            padding[1] = img->h - oldH;
        } else if (!strcmp(position, "rt")) {
            padding[0] = abs(img->w - oldW);
            padding[1] = img->h - oldH;
        } else if (!strcmp(position, "c")) {
            padding[0] = abs(img->w - oldW) / 2;
            padding[1] = (img->h - oldH) / 2;
        } else if (strcmp(position, "lb")) {
            fprintf(stderr, "Unknown position: %s.\n", position);
            exitFree(img);
        }

        Rgb **newPixs;
        if (oldW > img->w) {
            newPixs = _cutImg(img, oldW, oldH, color, padding);
        } else {
            newPixs = _expandImg(img, oldW, oldH, color, padding);
        }

        // Удаляем, чтобы не было утечек
        if (oldH > img->h) {
            for (int i = img->h; i < oldH; i++) {
                free(img->pixels[i]);
            }
        }
        img->pixels = realloc(img->pixels, img->h * sizeof(Rgb *));
        for (int i = 0; i < img->h; i++) {
               Rgb *tmp = realloc(img->pixels[i], img->w * sizeof(Rgb) +
(img->w * 3) % 4);
            if (!tmp) {
                fprintf(stderr, "You have written too big size of new
image!\n");
                // Не забываем очистить память в случае ошибки
                for (int j = i; j < img->h; j++) {
                    free(newPixs[j]);
                }
```

```
                free(newPixs);
                exitFree(img);
            }
            img->pixels[i] = tmp;
            for (int j = 0; j < img->w; j++) {
                img->pixels[i][j] = newPixs[i][j];
            }
            free(newPixs[i]);
        }
        free(newPixs);
    }


    void drawLine(Image *img, int *start, int *end, int thickness, char
*colorStr) {
        puts("Drawing the line...");
        start[1] = img->h - min(2, start[1], img->h) - 1;
        end[1] = img->h - min(2, end[1], img->h) - 1;
        // По алгоритму Брезенхема смещение относительно осей
        int deltaX = abs(end[0] - start[0]);
        int deltaY = abs(end[1] - start[1]);
        int signX = start[0] < end[0] ? 1 : -1;
        int signY = start[1] < end[1] ? 1 : -1;
        int error = deltaX - deltaY;
        Rgb color = getColor(colorStr, img);
        setPixels(img, end[0], end[1], thickness, color);
        while (start[0] != end[0] || start[1] != end[1]) {
            setPixels(img, start[0], start[1], thickness, color);
            int error2 = error * 2;
            if (error2 > -deltaY) {
                error -= deltaY;
                start[0] += signX;
            }
            if (error2 < deltaX) {
                error += deltaX;
                start[1] += signY;
            }
        }
```

```
    }

    unsigned char _inRound(int x0, int y0, int r, int x1, int y1) {
        return (x0-x1)*(x0-x1) + (y0-y1)*(y0-y1) <= r*r;
    }
    void setPixels(Image *img, int x, int y, int quan, Rgb color) {
        // Закрашиваем, если точка лежит в quan/2 окрестности (x, y)
        for (int i = max(2, y - quan / 2, 0); i < min(2, y + quan / 2 +
1, img->h); i++) {
            for (int j = max(2, x - quan / 2, 0); j < min(2, x + quan /
2 + 1, img->w); j++) {
                if (_inRound(x,y,quan/2,j,i)) {
                    img->pixels[i][j] = color;
                }
            }
        }
    }




    Rgb getColor(char *color, Image *img) {
        // Если пользователь ввел вместо цвета пробелы, будет черный
        int r=0, g=0, b=0;
        // FFFFFF <=> FFF
        if (strlen(color) == 6) {
            sscanf(color, "%02x%02x%02x", &r, &g, &b);
        } else if (strlen(color) == 3) {
            sscanf(color, "%01x%01x%01x", &r, &g, &b);
            r *= r;
            g *= g;
            b *= b;
        } else {
            fprintf(stderr, "Invalid color string %s.\n", color);
            exitFree(img);
        }
        Rgb res = {b, g, r};
```

```
    return res;
}


int min(int cnt,...) {
    va_list ap;
    int i, current, minimum;
    va_start(ap, cnt);
    minimum = INT_MAX;
    for (i = 0; i < cnt; i++){
        current = va_arg(ap, int);
        if (current < minimum)
            minimum = current;
    }
    va_end(ap);
    return minimum;
}

int max(int cnt,...) {
    va_list ap;
    int i, current, maximum;
    va_start(ap, cnt);
    maximum = INT_MIN;
    for (i = 0; i < cnt; i++){
        current = va_arg(ap, int);
        if (current > maximum)
            maximum = current;
    }
    va_end(ap);
    return maximum;
}
```