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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка BMP-изображений

Студентка гр. 2384 _____ Лавренова Ю.Д.

Преподаватель _____ Гаврилов А.В.

Санкт-Петербург
2023

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студентка Лавренова Ю.Д.

Группа 2384

Тема работы: Обработка BMP-изображений

Исходные данные:

Вариант № 5

Программа должна иметь CLI или GUI.

Общие сведения

- 24 бита на цвет
- без сжатия
- файл всегда соответствует формату BMP (но стоит помнить, что версий у формата несколько)
- обратите внимание на выравнивание; мусорные данные, если их необходимо дописать в файл для выравнивания, должны быть нулями.
- обратите внимание на порядок записи пикселей
- все поля стандартных BMP заголовков в выходном файле должны иметь те же значения что и во входном (разумеется кроме тех, которые должны быть изменены).

Программа должна реализовывать весь следующий функционал по обработке bmp-файла.

1. Инвертировать цвета в заданной окружности. Окружность определяется:
    - **либо** координатами левого верхнего и правого нижнего угла квадрата, в который она вписана, **либо** координатами ее центра и радиусом
2. Обрезка изображения. Требуется обрезать изображение по заданной области. Область определяется:
    - Координатами левого верхнего угла
    - Координатами правого нижнего угла

3. Рисование треугольника. Треугольник определяется
    - Координатами его вершин
    - Толщиной линий
    - Цветом линий
    - Треугольник может быть залит или нет
    - Цветом, которым он залит, если пользователем выбран залитый
4. Рисование отрезка. Отрезок определяется:
    - координатами начала
    - координатами конца
    - цветом
    - толщиной

Содержание пояснительной записки:

- аннотация
- содержание
- введение
- ход выполнения работы
- заключение
- список использованных источников
- приложения А и Б

Предполагаемый объем пояснительной записки: не менее 10 страниц.

Дата выдачи задания: 22.03.2022

Дата сдачи реферата: 20.05.2022

Дата защиты реферата: 22.05.2022

Студентка: гр. 2384 Лавренова Ю.Д.

Преподаватель: Гаврилов А.В.

АННОТАЦИЯ

Курсовая работа заключается в создании программы на языке программирования Си для обработки BMP-изображения. Программа работает только с версией файлов BMP3 и глубиной кодирования 24 бита на пиксель, а также без сжатия. Ввод информации осуществляется через CLI (Command Line Interface), в котором через команды опций и аргументов реализуется изменения изображения.

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ

Цель работы:

Необходимо написать программу на языке Си для работы с BMP-изображениями по запросам пользователя в рамках заявленного функционала варианта курсовой работы.

Задачи:

• реализовать чтение, создание BMP-изображения и хранение данных о нём

• реализовать функцию инвертирования цвета в заданной окружности

• реализовать обрезку изображения

• реализовать рисование треугольника, а также его заливки (на усмотрение пользователя)

• реализовать рисование отрезка

• реализовать чтение и обработку команд, введенных пользователем в терминале через CLI

2. ХОД ВЫПОЛНЕНИЯ РАБОТЫ

2.1. Работа с BMP-изображением.

Для хранения данных о BMP-изображении в файле *structs.h* программы реализовано 4 структуры. Первая - *BMPHeaderFile*. Она хранит заголовок файла: тип, размер, два резервных поля и положение данных о пикселях относительно начала заголовка файла. Вторая структура - *BMPHeaderInfo*. Она хранит остальную информацию об изображении: размер структуры файла, его высоту и ширину, количество слоёв, количество бит на пиксель, информацию о сжатии файла, размер матрицы изображения, количество пикселей на метр и сведения о цветовой таблице. У всех вышеупомянутых структур размер округляется до 1, с помощью *pragma pack()*. Третья структура — *RGB*. В ней реализовано хранение синего, зеленого и красного цвета палитры. Четвёртая — *BMP*. В ней хранятся указатели на структуры *BMPHeaderFile*, *BMPHeaderInfo*, имя (*char*) и двухмерный массив *RGB* пикселей.

Далее в файле *BMPwork.c* описаны функции для дальнейшей работы с BMP-файлом. Для чтения данных о BMP-изображении реализована функция *readBmp()*. Она принимает на вход имя файла и возвращает созданную *bmp_data*. Для начала функция открывает входящий файл(*fopen()*), а, если это невозможно сделать, выбрасывает исключение. Далее идет заполнение полей информации, а также заполнение массива пикселей для данного файла. В функции *saveBmp()* реализовано сохранение BMP-изображения в выходной файл: переписывание информации о файле, запись пикселей изображения. Функция *createBMP()* создает пустой BMP-файл, для дальнейшей записи. В функции реализовано заполнение информационных полей и заполнение массива пикселей. Функция *freeBMP()* реализовано очищение динамической памяти, выделенной для BMP-файла. Функция *print_bmp_info()* выводит всю необходимую информацию о BMP-изображении.

2.2 Проработка исключений и дополнительные функции

Реализованы функции для проработки исключений:

1) *print_not_enough_or_not_valid_args()* – функция выводит ошибку ввода недостаточного количества аргументов или их неверный тип, а также принудительно завершает программу.

2) *print_not_valid_args()* – функция выводит ошибку ввода неверного типа данных, а также принудительно завершает программу.

3) *check_color()* – функция проверяет соответствие цвета палитре RGB.

4) *maxi() и mini()* – функции соответственно выводят максимальное и минимальное значение из двух поданных.

2.3. Первая функция (*invertColorinCircle()*)

Данная функция описана в файле *function.c*

Для начала реализовано две вспомогательные функции:

1) *invetPixel()* – данная функция инвертирует цвет данного пикселя(подается координата в массиве): каждой компоненте цвета присваивается значение 255 – текущий цвет компоненты.

2) *checkIntoCircle()* – данная функция возвращает 1 – если данная точка находиться в заданной окружности(подается центр окружности и радиус), используя уравнение окружности, -1 – если неравенство не выполняется.

На вход функции *invertColorinCircle()* подается указатель на bmp-файл, а также массив аргументов. Далее происходит разыменование ключа –k, который отвечает за выбор того, как будет задана окружность:

1) k == 0:

Происходит разыменования аргументов - координат центра окружности и ее радиуса. Следующим, программа проходит во всем пикселям циклами *for(),* проверяет находиться ли точка в заданной области *heckIntoCircle(),* если это так, то вызывается функция *invetPixel().*

2) k == 1:

Происходит разыменования аргументов - координат верхнего левого и правого нижнего угла квадрата. Далее высчитывается центр окружности и ее радиус. Следующим, программа проходит во всем пикселям циклами *for(),* проверяет находиться ли точка в заданной области *heckIntoCircle(),* если это так, то вызывается функция *invetPixel().*

2.4. Вторая функция(cropimg())

Данная функция описана в файле *function.c*

Для реализации функции создана вспомогательная функция *createBMP(),* описанная в файле *BMPwork.c*. С начала происходит разыменования аргументов, поданных в функцию. Далее задается новая высота и ширина, которая подается в функцию *createBMP().* Далее записываются пиксели с помощью функции *memcpy().* Далее идет очищение массива пикселей и перезапись файла.

2.5. Третья функция(*drawTriangleFill()).*

Данная функция описана в файле *function.c*

В начале происходит разыменование аргументов функции, а также проверка значений на исключения. Далее в зависимости от того залит или нет треугольник, то используются разные вспомогательные функции:

1) залит

Вызывается функция *fillTriangle().* В ней находятся максимальные и минимальные значения для переменных x и y. Далее циклами *for()* программа проходится по заданной области. С помощью функции *checkPoint()* проверяется находится ли пиксель внутри треугольника, используя векторное произведение. Далее с помощью функции *paintPixel()* пикселю присваивается заданный цвет заливки по каждой компоненте цвета.

Вызывается функция *drawTriangle().* В ней вызывается рисование трех отрезков с помощью функции *drawLine_pr()* (описана ниже).

2) не залит

Вызывается функция *drawTriangle()*. В ней вызывается рисование трех отрезков с помощью функции *drawLine_pr()* (описана ниже).

2.6. Четвертая функция(*drawLine()*)

Данная функция описана в файле *function.c*

В начале функции происходит разыменование указателей аргументов, переданных в функцию, а также проверка этих значений. Далее вызывается вспомогательная функция *drawLine_pr()*. Используется алгоритм Брезентхема. Функция вначале высчитывает разницу, которую надо пойти по каждой из осей. Далее сравнивается начало и конец отрезка и присваивается знак перемещения по каждой из координат. Высчитывается переменная *error* и инициализируется *e2 = 0*. Далее циклом *while()* заполняется толщина пикселей и проверяется *error* и *e2*, соответственно изменяется смещение и текущие координаты. Цикл прерывается, когда текущие координаты станут равны конечным.

2.7. Реализация CLI

CLI реализован с использованием библиотеки *getopt.h*.

Имеется возможность вызова подсказки для работы с программой с помощью ключа *-h/--help*, либо запустив программу без ключей и аргументов (только *./a.out*).

Для реализации CLI:

1) была создана структура *Arguments* (в файле *structs.h*), в которой хранятся данные переданных аргументов

2) реализована функция *set_null()*, “зануляющая” значения аргументов

3) реализация функция *parse()*, которая разбивает строку вида “[число.число. …]” на несколько чисел и записывает их в нужные поля структуры *Arguments*

4) реализована функция *choice(),* которая в зависимости от переданного ей ключа заполняет нужные поля структуры *Arguments,* используя функции *parse()* и *atoi().*

5) реализована функция *functions_choice(),* которая в зависимости от ключа функции, которая записана в *prev_opt*, вызывает нужную функцию. Так как при попытке вызвать какую-либо функцию вначале пишется её ключ в переменную *opt,* а затем нужные ей ключи с аргументами, ключ данной функции надо сохранить в памяти в переменную *prev_opt* и вызвать её, когда будет считан ключ другой функции (перезапись *opt*), либо, когда закончится ввод в командной строке.

В самой функии *cli()* были инициализированы строка *opts* в которой хранятся все короткие ключи и структура *option*, в которой хранятся все длинные ключи в массиве *long_options*. Далее в цикле вызывается функция *getopt_long(),* пока она не вернёт -1 (конец ввода, переменная *opt == -1*).

В файле *main.c* вызывается функция *cli()* и передаются аргументы командной строки *argc* и *argv*, которые уже передаются в функцию *getopt_long().*

3. ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была реализована консольная программа по созданию и изменению BMP-изображений версии BMP3, без сжатия с стандартной RGB палитрой. Программа реализовывает запросы пользователя через CLI (Command Line Interface). В программе реализованы следующие функции: инвертирование изображение в заданной окружности, обрезка изображения, рисование и заливка треугольника и рисование линии. Для удобства пользователя программа может выводить информацию о файле, а также справочную информацию о ее работе.

4. СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Язык программирования Си: монография / Б.Керниган, Д.Ритчи; Пер. с англ. под ред. В.С.Штаркмана. - 3-е изд., испр. - СПб. : Невский диалект, 2001. -351 с

2. Интернет-сайт http://cppstudio.com

3. Интернет-сайт http://www.c-cpp.ru

ПРИЛОЖЕНИЕ А.
ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

1. Инвертирование цвета в заданной окружности.

Исходник:

Терминал:

```
ПРОБЛЕМЫ  ВЫХОДНЫЕ ДАННЫЕ  КОНСОЛЬ ОТЛАДКИ  ТЕРМИНАЛ  JUPYTER
julia@julia-VirtualBox:~/cw_2s$ make -f Makefile
gcc -c main.c
gcc -c cli.c
gcc -c functions.c
gcc -c BMPwork.c
gcc main.o cli.o functions.o BMPwork.o
rm -rf *.o
julia@julia-VirtualBox:~/cw_2s$ ./a.out ./tiger.bmp --invert -R 500.500 -r 300 -o ./output.bmp
julia@julia-VirtualBox:~/cw_2s$
```

Выход:

2. Обрезка изображения заданной области

Исходник:

Терминал:

```
ПРОБЛЕМЫ  ВЫХОДНЫЕ ДАННЫЕ  КОНСОЛЬ ОТЛАДКИ  ТЕРМИНАЛ  JUPYTER

julia@julia-VirtualBox:~/cw_2s$ make -f Makefile
gcc -c main.c
gcc -c cli.c
gcc -c functions.c
gcc -c BMPwork.c
gcc main.o cli.o functions.o BMPwork.o
rm -rf *.o
julia@julia-VirtualBox:~/cw_2s$ ./a.out ./shrek.bmp --crop -f 200.200 -s 700.700 -o ./output1.bmp
julia@julia-VirtualBox:~/cw_2s$
```

Выход: 

3. Рисование треугольника(залит)

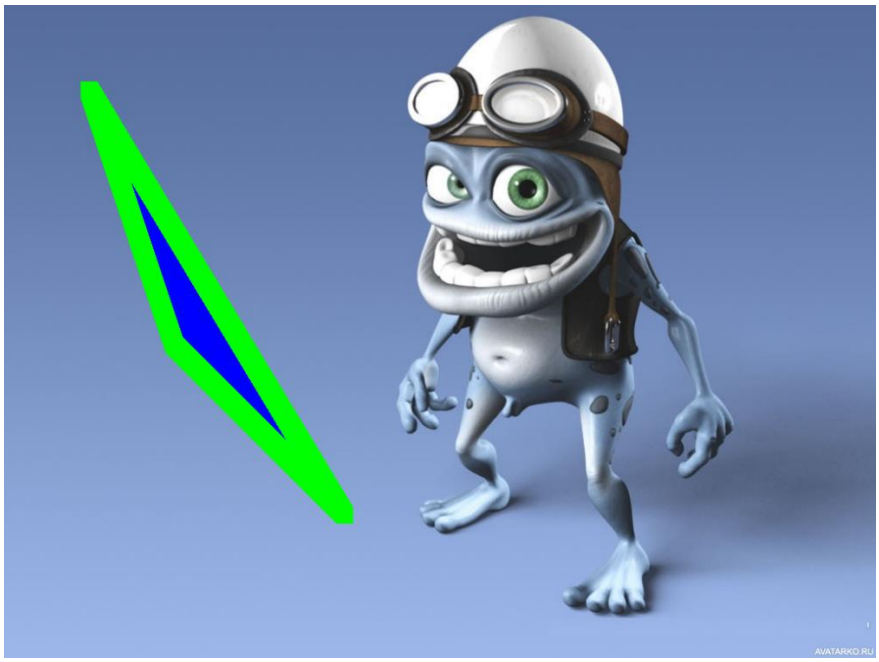
Исходник:

Терминал:

```
ПРОБЛЕМЫ  ВЫХОДНЫЕ ДАННЫЕ  КОНСОЛЬ ОТЛАДКИ  ТЕРМИНАЛ  JUPYTER

julia@julia-VirtualBox:~/cw_2s$ make -f Makefile
gcc -c main.c
gcc -c cli.c
gcc -c functions.c
gcc -c BMPwork.c
gcc main.o cli.o functions.o BMPwork.o
rm -rf *.o
julia@julia-VirtualBox:~/cw_2s$ ./a.out ./frog.bmp -T -f 100.100 -s 200.400 -t 400.600 -w 20 -c 0.255.0 -p 1 -F 255.0.0 -o ./output2.bmp
julia@julia-VirtualBox:~/cw_2s$
```

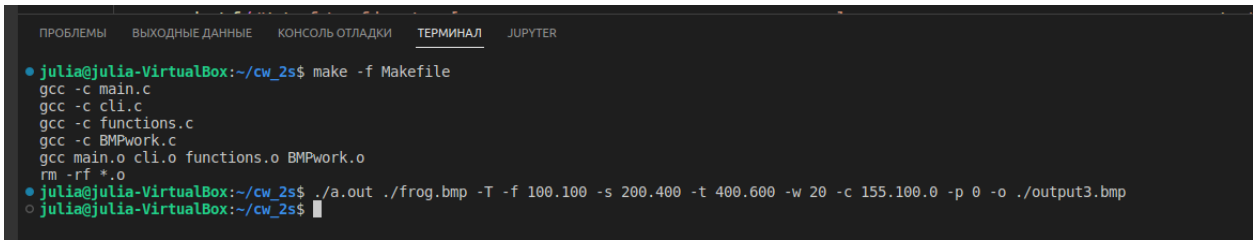
Выход:



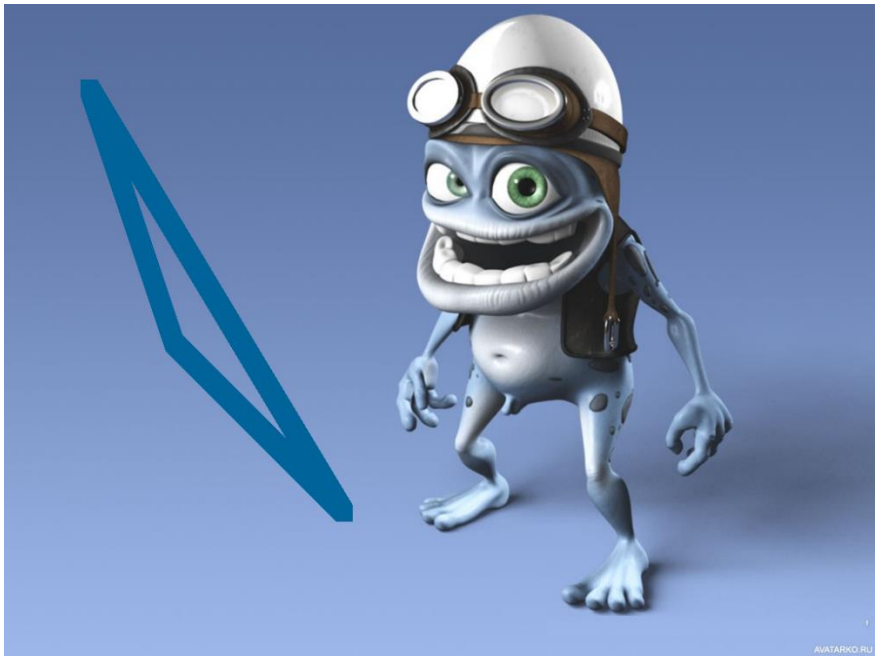
4. Рисование треугольника (не залит)

Исходник:

Терминал:

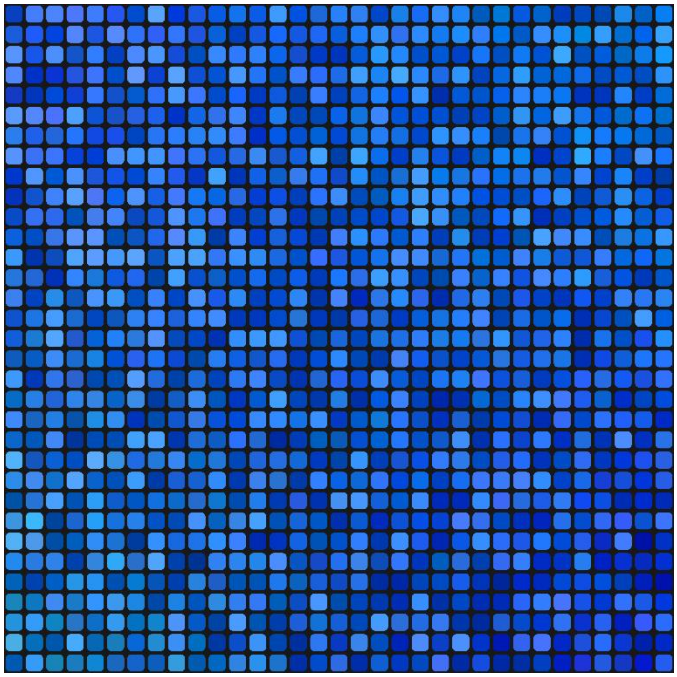


Выход:



5. Рисование отрезка

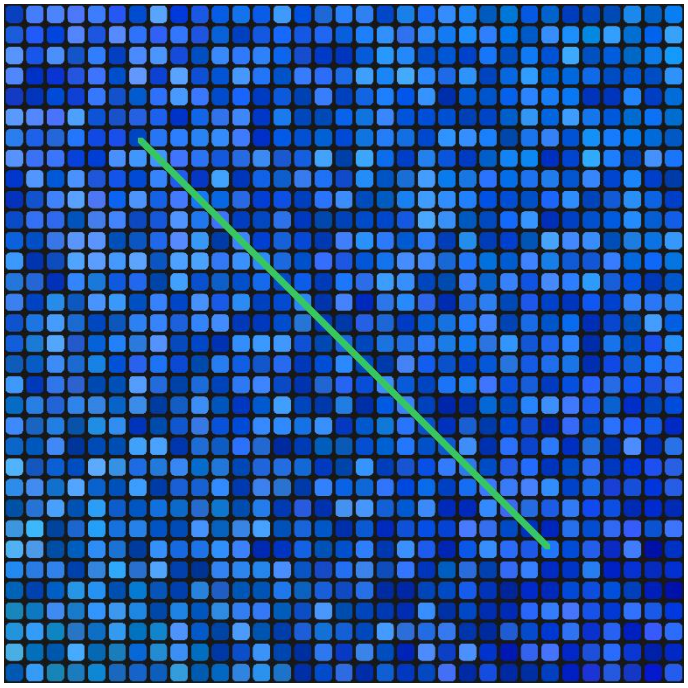
Исходник:



Терминал:

```
julia@julia-VirtualBox:~/cw_2s$ make -f Makefile
gcc -c main.c
gcc -c cli.c
gcc -c functions.c
gcc -c BMPwork.c
gcc main.o cli.o functions.o BMPwork.o
rm -rf *.o
julia@julia-VirtualBox:~/cw_2s$ ./a.out ./picture.bmp --line -f 1000.1000 -s 4000.4000 -w 35 -c 100.200.55 -o ./output5.bmp
julia@julia-VirtualBox:~/cw_2s$
```

Выход:



6. Выполнение нескольких функций

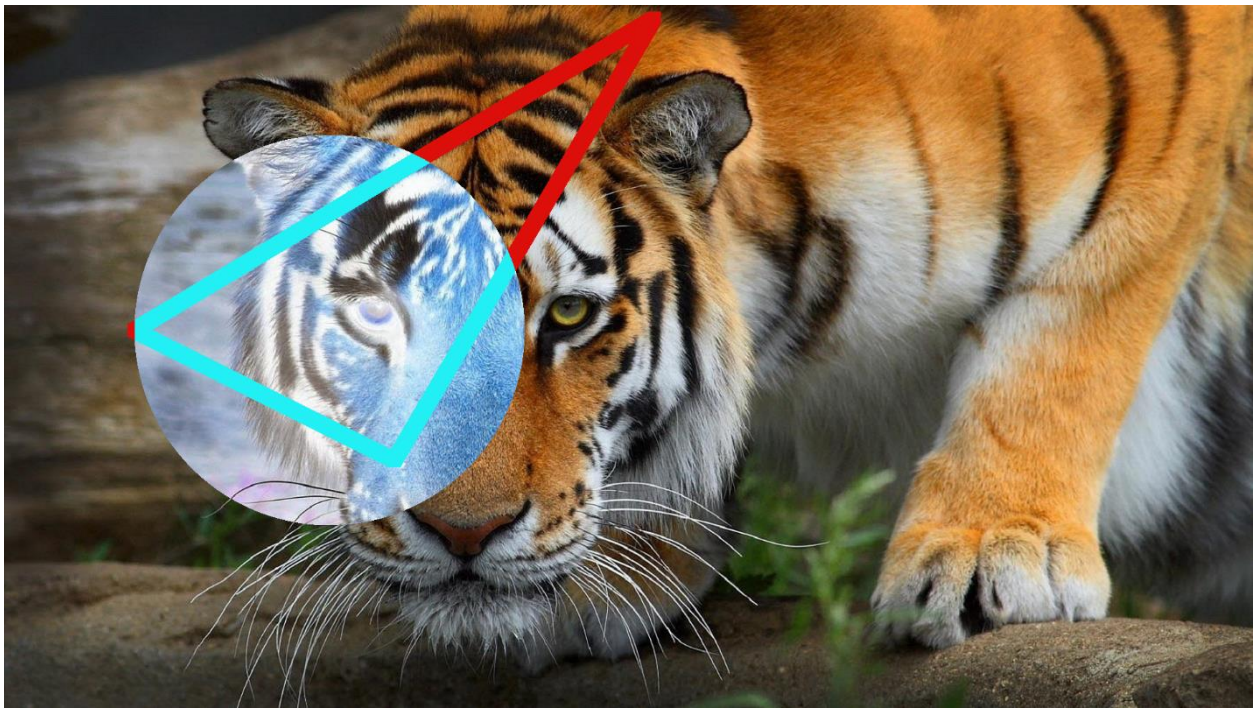
Исходник:

Терминал:

```
ПРОБЛЕМЫ  ВЫХОДНЫЕ ДАННЫЕ  КОНСОЛЬ ОТЛАДКИ  ТЕРМИНАЛ  JUPYTER

julia@julia-VirtualBox:~/cw_2s$ make -f Makefile
gcc -c main.c
gcc -c cli.c
gcc -c functions.c
gcc -c BMPwork.c
gcc main.o cli.o functions.o BMPwork.o
rm -rf *.o
julia@julia-VirtualBox:~/cw_2s$ ./a.out ./tiger.bmp --triangle -f 1000.20 -s 200.500 -t 600.700 -w 20 -c 10.15.220 -p 0 --invert -R 500.500 -r 300 -o ./output6.bmp
julia@julia-VirtualBox:~/cw_2s$
```

Выход:



ПРИЛОЖЕНИЕ Б.
ИСХОДНЫХ КОД ПРОГРАММЫ.

Название файла: main.c

```
#include "headers.h"
#include "cli.h"
#include "functions.h"
#include "BMPwork.h"

int main (int argc, char **argv){
    cli(argc,argv);
    return 0;
}
```

Название файла: headers.h

```
#pragma once
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <getopt.h>
```

Название файла: structs.h

```
#pragma once
#include "headers.h"

#pragma pack(push, 1)
typedef struct {
    unsigned short bfType;
    unsigned int bfSize;
    unsigned short bfReserved1;
    unsigned short bfReserved2;
    unsigned int bfBits;
} BMPHeaderFile;

typedef struct{
    unsigned int headerSize;
```

```
    int width;
    int height;
    unsigned short planes;
    unsigned short bitperpixel;
    unsigned int compr;
    unsigned int imgSize;
    unsigned int x_pelspermetr;
    unsigned int y_pelspermetr;
    unsigned int clr_table;
    unsigned int clr_important;
} BMPHeaderInfo;

typedef struct{
    unsigned char b;
    unsigned char g;
    unsigned char r;
} RGB;

typedef struct{
    BMPHeaderFile BMPH;
    BMPHeaderInfo BMPH_inf;
    RGB **img;
    char *name;
} BMP;

typedef struct arguments{
    int x1, y1;
    int x2, y2;
    int x3, y3;
    int x, y;
    int width;
    int color_r, color_g, color_b;
    int fill_r, fill_g, fill_b;
    int radius;
    int pour;
    char* output;
    int k;
} arguments;
```

```
#pragma pack(pop)
```

Название файла: BMPwork.c

```
#include "headers.h"
#include "structs.h"
#include "BMPwork.h"

BMP readBmp(const char* file_name){
    FILE* fp = fopen(file_name, "rb");

    if(!fp){
        printf("Не удалось открыть файл для выполения.\n");
        exit(-1);
    }

    BMP bmp_data;
    fread(&bmp_data.BMPH, 1, sizeof(bmp_data.BMPH), fp);
    fread(&bmp_data.BMPH_inf, 1, sizeof(bmp_data.BMPH_inf), fp);

    if(bmp_data.BMPH_inf.bitperpixel != 24){
        printf("Программа не обрабатывает файлы, у которых не выделено
24 бита на цвет.\n");
        exit(-1);
    }

    size_t height = bmp_data.BMPH_inf.height;
    size_t width = bmp_data.BMPH_inf.width;

    int padding = (width * sizeof(RGB)) % 4;
    if (padding != 0) padding = 4 - padding;

    bmp_data.img = (RGB**)malloc(sizeof(RGB*)*height);
    for (size_t i = 0; i < height; i++){
        bmp_data.img[i] = (RGB*)malloc(sizeof(RGB)*width + padding);
        fread(bmp_data.img[i], 1, sizeof(RGB)*width +padding, fp);
    }

    fclose(fp);
```

```
    return bmp_data;
}
//save
void saveBmp(const char* file_name, BMP bmp_data){
    FILE* fp = fopen(file_name, "wb");
    fwrite(&bmp_data.BMPH, 1, sizeof(bmp_data.BMPH), fp);
    fwrite(&bmp_data.BMPH_inf, 1, sizeof(bmp_data.BMPH_inf), fp);

    size_t width = bmp_data.BMPH_inf.width;
    size_t height = bmp_data.BMPH_inf.height;

    int padding = (width * sizeof(RGB)) % 4;
    if (padding != 0) padding = 4 - padding;

    for (size_t i = 0; i < height; i++){
        fwrite(bmp_data.img[i], 1, sizeof(RGB)*width + padding, fp);
    }
    fclose(fp);
}
void print_bmp_info(BMP* bmp_data){
    printf("Информация о bmp-файле:\n");
    printf("file type = \t%hi\n", bmp_data->BMPH.bfType);
    printf("file size = \t%hi\n", bmp_data->BMPH.bfSize);
    printf("reserved1 = \t%hi\n", bmp_data->BMPH.bfReserved1);
    printf("reserved2 = \t%hi\n", bmp_data->BMPH.bfReserved2);
    printf("size of bits = \t%hi\n", bmp_data->BMPH.bfBits);
    printf("bit per pixel = \t%hi\n", bmp_data->BMPH_inf.bitperpixel);
    printf("size of header = \t%hi\n", bmp_data->BMPH_inf.headerSize);
    printf("height = \t%d\n", bmp_data->BMPH_inf.height);
    printf("width = \t%d\n", bmp_data->BMPH_inf.width);
    printf("size of img = \t%hi\n", bmp_data->BMPH_inf.imgSize);
    printf("important color count = \t%hi\n",
bmp_data->BMPH_inf.clr_important);
    printf("colors in color table = \t%hi\n",
bmp_data->BMPH_inf.clr_table);
    printf("compression = \t%hi\n", bmp_data->BMPH_inf.compr);
    printf("planes = \t%hi\n", bmp_data->BMPH_inf.planes);
```

```
    printf("x pixels per meter = \t%hi\n",
bmp_data->BMPH_inf.x_pelspermetr);
    printf("y pixels per meter = \t%hi\n",
bmp_data->BMPH_inf.y_pelspermetr);

}

BMP createBMP(size_t width, size_t height){
    BMP bmp_data;
    //file
    int padding = (width * sizeof(RGB)) % 4;
    if (padding != 0) padding = 4 - padding;
    bmp_data.BMPH.bfType = 0x4D42;
    bmp_data.BMPH.bfSize = height*(width*sizeof(RGB) + padding) +
sizeof(BMPHeaderFile)+sizeof(BMPHeaderInfo);
    bmp_data.BMPH.bfReserved1 = 0;
    bmp_data.BMPH.bfReserved2 = 0;
    bmp_data.BMPH.bfBits =
sizeof(BMPHeaderFile)+sizeof(BMPHeaderInfo);

    //info
    bmp_data.BMPH_inf.headerSize = 40;
    bmp_data.BMPH_inf.width = width;
    bmp_data.BMPH_inf.height = height;
    bmp_data.BMPH_inf.planes = 1;
    bmp_data.BMPH_inf.bitperpixel = sizeof(RGB)*8;
    bmp_data.BMPH_inf.compr = 0;
    bmp_data.BMPH_inf.imgSize = height*(width*sizeof(RGB) + padding);
    bmp_data.BMPH_inf.x_pelspermetr = 0;
    bmp_data.BMPH_inf.y_pelspermetr = 0;
    bmp_data.BMPH_inf.clr_important = 0;
    bmp_data.BMPH_inf.clr_table = 0;

    bmp_data.img = (RGB**)malloc(sizeof(RGB*)*height);
    for (size_t i = 0; i < height; i++){
        bmp_data.img[i] = (RGB*)malloc(sizeof(RGB)*width + padding);
        for (size_t j = 0; j < width; j++){
            bmp_data.img[i][j].b = 255;
```

```
            bmp_data.img[i][j].g = 255;
            bmp_data.img[i][j].r = 255;
        }
    }
    return bmp_data;
}

void freeBMP(BMP* bmp_file) {
    if (bmp_file) {
        for(int i = 0; i < bmp_file->BMPH_inf.height; i++) {
            free(bmp_file->img[i]);
        }
        free(bmp_file->img);
    }
}
```

Название файла: BMPwork.h

```
#pragma once
#include "structs.h"

BMP readBmp(const char* file_name);
void saveBmp(const char* file_name, BMP bmp_data);
void print_bmp_info(BMP* bmp_data);
BMP createBMP(size_t width, size_t height);
void freeBMP(BMP* bmp_file);
```

Название файла: functions.c

```
#include "headers.h"
#include "structs.h"
#include "BMPwork.h"

void print_not_enough_or_not_valid_args() {
    printf("Было введено недостаточное количество аргументов, либо
данные не являются валидными. Работа программы принудительна
завершена.\n");
    exit(-1);
}
```

```
void print_not_valid_args() {
    printf("Данные аргументов не являются валидными. Работа программы
принудительна завершена.\n");
    exit(-1);
}

int check_color(int color) {
    return !(0 <= color && color <= 255);
}

int maxi(int a, int b){
    if (a > b) return a;
    else return b;
}
int mini(int a, int b){
    if (a < b) return a;
    else return b;
}

//drawpix
void paintPixel(BMP* bmp_data, int x, int y, RGB color){
    if (x < 0 || y < 0 || x >= bmp_data->BMPH_inf.width || y >=
bmp_data->BMPH_inf.height){
        return;
    }
    else{
        bmp_data->img[y][x].b = color.b;
        bmp_data->img[y][x].g = color.g;
        bmp_data->img[y][x].r = color.r;
    }
}

//drawline
void drawLine_pr(BMP* bmp_data, int x1, int y1, int x2, int y2, int
wid, RGB color){
    y1 = bmp_data->BMPH_inf.height - y1;
    y2 = bmp_data->BMPH_inf.height - y2;
```

```
    int dx = abs(x2 - x1);
    int dy = abs(y2 - y1);
    int sx, sy;
    if (x1 < x2) sx = 1;
    else sx = -1;
    if (y1 < y2) sy = 1;
    else sy = -1;
    int err = dx - dy;
    int e2 = 0;
    while (1) {
        for (int i = 0; i < wid; i++) {
            for (int j = 0; j < wid; j++) {
                int x = x1 - (wid / 2) + i;
                int y = y1 - (wid / 2) + j;
                paintPixel(bmp_data, x, y, color);
            }
        }
        if (x1 == x2 && y1 == y2) {
            break;
        }
        e2 = 2 * err;
        if (e2 > -dy) {
            err -= dy;
            x1 += sx;
        }
        if (e2 < dx) {
            err += dx;
            y1 += sy;
        }
    }
}

void drawLine(BMP* bmp, arguments* args){
    int *x1 = &(args->x1);
    int *y1 = &(args->y1);
    int *x2 = &(args->x2);
    int *y2 = &(args->y2);
```

```
    int *wid = &(args->width);
    int *c_r = &(args->color_r);
    int *c_b = &(args->color_b);
    int *c_g = &(args->color_g);
    RGB color = {*c_b, *c_g, *c_r};

    if (*x1 < 0 || *x2 < 0 || *y1 < 0 || *y2 < 0 || *wid <= 0){
        print_not_valid_args();
        exit(-1);
    }

    if (*x1 > bmp->BMPH_inf.width || *x2 > bmp->BMPH_inf.width ||
*y1 > bmp->BMPH_inf.height || *y2 > bmp->BMPH_inf.height){
        print_not_valid_args();
        exit(-1);
    }

    if (check_color(*c_b) || check_color(*c_g) || check_color(*c_r)){
        print_not_enough_or_not_valid_args();
        exit(-1);
    }

    drawLine_pr(bmp, *x1, *y1, *x2, *y2, *wid, color);
    *x1= -1; *x2 = -1; *y1 = -1; *y2 = -1; *wid = -1; *c_b = -1; *c_g
= -1; *c_r = -1;
}
//triangle
void drawTriangle(BMP* bmp, int x1, int y1, int x2, int y2, int x3,
int y3, int wid, RGB color){
    drawLine_pr(bmp, x1, y1, x2, y2, wid, color);
    drawLine_pr(bmp, x2, y2, x3, y3, wid, color);
    drawLine_pr(bmp, x1, y1, x3, y3, wid, color);
}

//fillTriangle
int checkPoint(int x, int y, int x1, int x2, int x3, int y1, int y2,
int y3){
    int a1,a2,a3;
```

```
    a1 = (x1-x)*(y2-y1) - (x2-x1)*(y1-y);
    a2 = (x2-x)*(y3-y2) - (x3-x2)*(y2-y);
    a3 = (x3-x)*(y1-y3) - (x1-x3)*(y3-y);
    if ( (a1 < 0 && a2 < 0 && a3 < 0 ) || (a1 > 0 && a2 > 0 && a3 >
0)) return 1;
    return 0;
}

void fillTriangle(BMP* bmp, int x1, int y1, int x2, int y2, int x3,
int y3, RGB fill){
    y1 = bmp->BMPH_inf.height - y1;
    y2 = bmp->BMPH_inf.height - y2;
    y3 = bmp->BMPH_inf.height - y3;
    int x_max = maxi(x1, maxi(x2,x3));
    int x_min = mini(x1, mini(x2,x3));
    int y_max = maxi(y1, maxi(y2,y3));
    int y_min = mini(y1, mini(y2,y3));
    for (int i = x_min; i <= x_max; i++){
        for (int j = y_min; j <= y_max; j++){
            if (checkPoint(i,j,x1,x2,x3,y1,y2,y3) == 1){
                paintPixel(bmp, i, j, fill);
            }
        }
    }
}

void drawTriangleFill(BMP* bmp, arguments* args){
    int* x1 = &(args->x1);
    int* y1 = &(args->y1);
    int* x2 = &(args->x2);
    int* y2 = &(args->y2);
    int* x3 = &(args->x3);
    int* y3 = &(args->y3);
    int* wid = &(args->width);
    int *c_r = &(args->color_r);
    int *c_b = &(args->color_b);
    int *c_g = &(args->color_g);
    RGB color = {*c_b, *c_g, *c_r};
```

```
    int* pouring = &(args->pour);
    int *f_r = &(args->fill_r);
    int *f_b = &(args->fill_b);
    int *f_g = &(args->fill_g);
    RGB fill = {*f_b, *f_g, *f_r};

    if (*x1 < 0 || *x2 < 0 || *x3 < 0 || *y1 < 0 || *y2 < 0 || *y3 < 0
|| (*pouring != 0 && *pouring != 1) || *wid <= 0){
        print_not_valid_args();
        exit(-1);
    }

    if (*x1 > bmp->BMPH_inf.width || *x2 > bmp->BMPH_inf.width ||
*x3 > bmp->BMPH_inf.width || *y1 > bmp->BMPH_inf.height || *y2 >
bmp->BMPH_inf.height || *y3 > bmp->BMPH_inf.height){
        print_not_valid_args();
        exit(-1);
    }

    if (check_color(*c_b) || check_color(*c_g) || check_color(*c_r)){
        print_not_enough_or_not_valid_args();
        exit(-1);
    }

    if (*pouring == 1 && (check_color(*f_b) || check_color(*f_g) ||
check_color(*f_r))){
        print_not_enough_or_not_valid_args();
        exit(-1);
    }

    if (*pouring){
        fillTriangle(bmp, *x1, *y1, *x2, *y2, *x3, *y3, fill);
        drawTriangle(bmp, *x1, *y1, *x2, *y2, *x3, *y3, *wid, color);
    }
    else drawTriangle(bmp, *x1, *y1, *x2, *y2, *x3, *y3, *wid, color);
}
//crop
void cropimg(BMP* bmp, arguments* args){
```

```
    int* x1 = &(args->x1);
    int *x2 = &(args->x2);
    int* y1 = &(args->y1);
    int* y2 = &(args->y2);

    if (*x1 < 0 || *x2 < 0 || *y1 < 0 || *y2 < 0 || *x1 >
bmp->BMPH_inf.width || *x2 > bmp->BMPH_inf.width || *y1 >
bmp->BMPH_inf.height || *y2 > bmp->BMPH_inf.height){
        print_not_valid_args();
        exit(-1);
    }

    int new_width = abs(*x1-*x2);
    int new_heigth = abs(*y1-*y2);
    int x_min = mini(*x1,*x2);
    *y1 = bmp->BMPH_inf.height - *y1  - 1;
    *y2 = bmp->BMPH_inf.height - *y2  - 1;
    int y_min = mini(*y1,*y2);

    BMP new_bmp = createBMP(new_width, new_heigth);

    int padding = (new_width * sizeof(RGB)) % 4;
    if (padding != 0) padding = 4 - padding;

    for (int i = 0; i < new_heigth; i++){
        memcpy(new_bmp.img[i], bmp->img[y_min + i] + x_min,
sizeof(RGB)*new_width + padding);
    }
    freeBMP(bmp);
    (*bmp) = new_bmp;
    *x1 = -1; *x2 = -1; *y1 = -1; *y2 = -1;
}

//invert
int checkIntoCircle(int x, int y, int x0, int y0, int r){
    if ((x - x0)*(x - x0) + (y - y0)*(y - y0) < r*r) return 1;
    else return -1;
}
```

```
void invertPixel(BMP* bmp, int x, int y){
    RGB color;
    color.b = 255 - bmp->img[y][x].b;
    color.g = 255 - bmp->img[y][x].g;
    color.r = 255 - bmp->img[y][x].r;
    paintPixel(bmp, x, y, color);
}

void invertColorinCircle(BMP* bmp, arguments* args){
    int *k = &(args->k);
    if (*k){
        int* x1 = &(args->x1);
        int* x2 = &(args->x2);
        int* y1 = &(args->y1);
        int* y2 = &(args->y2);
        int x = abs(*x1-*x2)/2 + *x1;
        int y = abs(*y1-*y2)/2 + *y1;
        int r = abs(*x1-*x2)/2;
        if (x < 0 || y < 0 || x >= bmp->BMPH_inf.width || y >=
bmp->BMPH_inf.height || r <= 0 || r > bmp->BMPH_inf.height || r >
bmp->BMPH_inf.width){
            print_not_valid_args();
            exit(-1);
        }
        int x_n = x - r;
        int x_k = x + r;
        y = bmp->BMPH_inf.height - y;
        int y_n = y - r;
        int y_k = y + r;
        for (int i = x_n; i <= x_k; i++){
            for (int j = y_n; j <= y_k; j++){
                if (checkIntoCircle(i,j,x,y,r) ==  1){
                    invertPixel(bmp, i, j);
                }
            }
        }
```

```
        *x1 = -1; *y1 = -1; *x2 = -1; *y2 = -1;
    }
    else{
        int* x = &(args->x);
        int* y = &(args->y);
        int* r = &(args->radius);
        if (*x < 0 || *y < 0 || *x >= bmp->BMPH_inf.width || *y >=
bmp->BMPH_inf.height || *r <= 0 || *r > bmp->BMPH_inf.height || *r >
bmp->BMPH_inf.width){
            print_not_valid_args();
            exit(-1);
        }
        int x1 = *x - *r;
        int x2 = *x + *r;
        *y = bmp->BMPH_inf.height - *y;
        int y1 = *y - *r;
        int y2 = *y + *r;

        for (int i = x1; i <= x2; i++){
            for (int j = y1; j <= y2; j++){
                if (checkIntoCircle(i,j,*x,*y,*r) ==  1){
                    invertPixel(bmp, i, j);
                }
            }
        }

        *x = -1; *y = -1; *r = -1;
    }
}
```

Название файла: functions.h

```
#pragma once
#include "headers.h"
#include "BMPwork.h"

int maxi(int a, int b);
int mini(int a, int b);
void paintPixel(BMP* bmp_data, int x, int y, RGB color);
```

```
void drawLine_pr(BMP* bmp_data, int x1, int y1, int x2, int y2, int
wid, RGB color);
void drawLine(BMP* bmp_data, arguments* args);
void drawTriangle(BMP* bmp, int x1, int y1, int x2, int y2, int x3,
int y3, int wid, RGB color);
int checkPoint(int x, int y, int x1, int x2, int x3, int y1, int y2,
int y3);
void fillTriangle(BMP* bmp, int x1, int y1, int x2, int y2, int x3,
int y3, RGB fill);
void drawTriangleFill(BMP* bmp, arguments* args);
void cropimg(BMP* bmp, arguments* args);
int checkIntoCircle(int x, int y, int x0, int y0, int r);
void invertPixel(BMP* bmp, int x, int y);
void invertColorinCircle(BMP* bmp, arguments* args);
```

Название файла: cli.c

```
#include "headers.h"
#include "structs.h"
#include "BMPwork.h"
#include "functions.h"

#pragma pack(push, 1)
const struct option long_options[] = {
    {"invert", no_argument, NULL, 'I'},
    {"crop", no_argument, NULL, 'C'},
    {"triangle", no_argument, NULL, 'T'},
    {"line", no_argument, NULL, 'L'},

    {"first", required_argument, NULL, 'f'},
    {"second", required_argument, NULL, 's'},
    {"third", required_argument, NULL, 't'},
    {"ring", required_argument, NULL, 'R'},
    {"radius", required_argument, NULL, 'r'},
    {"width", required_argument, NULL, 'w'},
    {"color", required_argument, NULL, 'c'},
    {"pouring", required_argument, NULL, 'p'},
    {"fill", required_argument, NULL, 'F'},
    {"square", required_argument, NULL, 'k'},
```

```
    {"output", required_argument, NULL, 'o'},
    {"help", no_argument, NULL, 'h'},
    {"info", no_argument, NULL, 'i'},
    {NULL, no_argument, NULL, 0}
};
#pragma pack(pop)

void print_help(){
    printf("Руководство по использованию программы:\n");
    printf("Программа обрабатывает BMP-файлы версии BMP3. С глубиной изображения 24 бита на пиксель\n");
    printf("Формат ввода: [имя исполняемого файла] [имя BMP-файла для обработки] [функция] -[ключ]/--[полный ключ] [аргументы и их ключи]...\n");
    printf("Функции/ключи:\n");

    printf("-I/--invert -- инвертирование цвета в заданной окружности. \n");
    printf("\t-k/--square -- [1(true)/0(false)] - задана окружность квадратом(true) или задана центром и радиусом окружности(false).\n ");
    printf("\t-f/--first [<x-координата>.<y-координата>] - координата левого верхнего угла квадрата.\n");
    printf("\t-s/--second [<x-координата>.<y-координата>] - координата правого нижнего угла квадрата.\n");
    printf("\t-R/--ring [<x - координата>.<y - координата>] - координаты цента окружности.\n");
    printf("\t-r/--radius [число]- длина радиуса окружности.\n");
```

```
    printf("-C/--crop -- обрезка изображения по заданной области.\n");
    printf("\t-f/--first  [<x-координата>.<y-координата>] - координата левого верхнего угла области.\n");
    printf("\t-s/--second [<x-координата>.<y-координата>] - координата правого нижнего угла области.\n");

    printf("-T/--triangle -- рисование треугольника.\n");
    printf("\t-f/--first  [<x-координата>.<y-координата>] - координата первой вершины.\n");
    printf("\t-s/--second [<x-координата>.<y-координата>] - координата второй вершины.\n");
    printf("\t-t/--third [<x-координата>.<y-координата>] - координата третьей вершины.\n");
    printf("\t-w/--width - [число] толщина линий треугольника.\n");
    printf("\t-c/--color - [число.число.число] цвет линий.\n");
    printf("\t-p/--pouring [1(true)/0(false)] - залит треугольник или нет\n");
    printf("\t-F/--fill - [число.число.число] цвет заливки треугольника.\n");

    printf("-L/--line -- рисование отрезка.\n");
    printf("\t-f/--first  [<x-координата>.<y-координата>] - координата начала отрезка.\n");
    printf("\t-s/--second [<x-координата>.<y-координата>] - координата конца отрезка.\n");
    printf("\t-w/--width [число] - толщина линии.\n");
    printf("\t-c/--color [число.число.число] - цвет линии.\n");
```

```
    printf("-h/--help -- справка о работе программ
ы.\n");
    printf("-i/--info -- информация о bmp-файле.\n");
    printf("-o/--output -- файл для записи полученного
изображения.\n");
}

arguments* set_null(arguments* args){
    args->color_b = -1; args->color_g = -1; args->color_r = -1;
    args->fill_b = -1; args->fill_g = -1; args->fill_r = -1;
    args->pour = -1;
    args->k = -1;
    args->radius = -1;
    args->width = -1;
    args->x1 = -1; args->x2 = -1; args->x3 = -1;
    args->y1 = -1; args->y2 = -1; args->y3 = -1;
    args->x = -1; args->y = -1;
    args->output = NULL;
    return args;
}

void parse_str(int** arr, int count, char* optarg){
    char* istr = strtok(optarg, ".");
    *(arr[0]) = atoi(istr);
    for (int i = 1; i < count; i++){
        istr = strtok(NULL, ".");
        if (istr != NULL) *(arr[i]) = atoi(istr);
        else {
            printf("Введено надостаточно аргументов
для реализации функции. Работа программы зав
ершена.\n");
            exit(-1);
        }
    }
}

arguments* choice(arguments *args, int opt){
```

```
int** arr = malloc(3*sizeof(int*));

switch (opt){
    case 'R':
        arr[0] = &(args->x);
        arr[1] = &(args->y);
        parse_str(arr, 2, optarg);
        break;
    case 'f':
        arr[0] = &(args->x1);
        arr[1] = &(args->y1);
        parse_str(arr, 2, optarg);
        break;
    case 's':
        arr[0] = &(args->x2);
        arr[1] = &(args->y2);
        parse_str(arr, 2, optarg);
        break;
    case 't':
        arr[0] = &(args->x3);
        arr[1] = &(args->y3);
        parse_str(arr, 2, optarg);
        break;
    case 'w':
        args->width = atoi(optarg);
        break;
    case 'r':
        args->radius = atoi(optarg);
        break;
    case 'p':
        args->pour = atoi(optarg);
        break;
    case 'c':
        arr[0] = &(args->color_b);
        arr[1] = &(args->color_g);
        arr[2] = &(args->color_r);
        parse_str(arr, 3, optarg);
        break;
```

```
        case 'F':
            arr[0] = &(args->fill_b);
            arr[1] = &(args->fill_g);
            arr[2] = &(args->fill_r);
            parse_str(arr, 3, optarg);
            break;
        case 'o':
            args->output = malloc(strlen(optarg)+1);
            strcpy(args->output, optarg);
            break;
        case 'h':
            print_help();
            break;
        case 'k':
            args->k = atoi(optarg);
            break;
        default:
            break;
        }
    free(arr);
    return args;
}

BMP func_choice(BMP file, int opt, int prev_opt, arguments* args){
    if (opt == 'I' || opt == 'L' || opt == 'C' || opt == 'T'){
        switch (prev_opt){
        case 'I':
            invertColorinCircle(&file, args);
            break;
        case 'L':
            drawLine(&file, args);
            break;
        case 'C':
            cropimg(&file, args);
            break;
        case 'T':
            drawTriangleFill(&file, args);
            break;
```

```
        default:
            break;
        }
    }
    return file;
}

void cli(int argc, char** argv){
    char *opts = "ICTLf:s:t:k:R:r:w:c:p:F:hio:";
    arguments *args = malloc(sizeof(arguments));
    args = set_null(args);
    int indexes;
    char* file_name = argv[1];
    int opt;
    int prev_opt = 'N';

    if (argc == 1){
        print_help();
        exit(-1);
    }

    BMP bmp_file = readBmp(file_name);
    args->output = file_name;

    opt = getopt_long(argc, argv, opts, long_options, &indexes);
    while (opt != -1){
        bmp_file = func_choice(bmp_file, opt, prev_opt, args);
        if (opt == 'I' || opt == 'L' || opt == 'C' || opt == 'T')
prev_opt = opt;
        if (opt == 'i') print_bmp_info(&bmp_file);

        args = choice(args ,opt);

        opt = getopt_long(argc, argv, opts, long_options, &indexes);
    }
    bmp_file = func_choice(bmp_file, 'I', prev_opt, args);
    saveBmp(args->output, bmp_file);
}
```

Название файла: cli.h

```
#pragma once
#include "structs.h"

void print_help();
arguments* set_null(arguments* args);
void parse_str(int** arr, int count, char* optarg);
arguments* choice(arguments *args, int opt);
void func_choice(BMP* file, int opt, int prev_opt, arguments* args);
void cli(int argc, char** argv);
```

Название файла: Makefile

```
all: main clean

main: main.o cli.o functions.o BMPwork.o
	gcc main.o cli.o functions.o BMPwork.o

main.o: main.c
	gcc -c main.c

functions.o: functions.c
	gcc -c functions.c

cli.o: cli.c
	gcc -c cli.c

BMPwork.o: BMPwork.c
	gcc -c BMPwork.c

clean:
	rm -rf *.o
```