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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка BMP-файлов

Студентка гр. 0382 _____ Рубежова Н.А.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2021

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студентка Рубежова Н.А.

Группа 0382

Тема работы: Обработка BMP-файлов
Вариант 4

Исходные данные:

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. Инверсия цвета в заданной области. Функционал определяется
    - Координатами левого верхнего угла области
    - Координатами правого нижнего угла области

2. Преобразовать в Ч/Б изображение (любым простым способом). Функционал определяется
    - Координатами левого верхнего угла области
    - Координатами правого нижнего угла области
3. Изменение размера изображения с его обрезкой или расширением фона. Функционал определяется:
    - Действием: увеличение или уменьшение изображения
    - Цветом фона при увеличении изображения
    - Точкой относительно которой производится действие: центр, левый верхний, правый верхний, левый нижний, правый нижний угол
4. Рисование отрезка. Отрезок определяется:
    - координатами начала
    - координатами конца
    - цветом
    - толщиной

Предполагаемый объем пояснительной записки:

Не менее 11 страниц.

Дата выдачи задания: 05.04.2021

Дата сдачи реферата: 30.04.2021

Дата защиты реферата: 01.06.2021

Студентка _____ Рубежова Н.А.

Преподаватель _____ Берленко Т.А.

АННОТАЦИЯ

В ходе выполнения курсовой работы создавалась программа на языке C, которая обрабатывает BMP-файл. Программа имеет CLI(Command Line Interface) с возможностью вывода справки о программе, реализуемых функциях и ключах. Программа поддерживает BMP-файлы версии V3 с BITMAPINFOHEADER, занимающим 40 байт, при этом глубина кодирования 24 бита, сжатия нет. Разработка велась на операционной системе Linux в IDE CLion с использованием компилятора gcc.

Инструкция по запуску: скомпилировать файл main.c и запустить исполняемый файл a.out.

SUMMARY

In the course work a C program was created that processes a BMP file. The program has a CLI (Command Line Interface) with the ability to display help about the program, implemented functions and keys. The program supports BMP files of version V3 with BITMAPINFOHEADER, which occupies 40 bytes, with a coding depth of 24 bits, no compression. Development was carried out on the Linux operating system in the CLion IDE using the gcc compiler.

How to start: compile the main.c file and run the a.out executable file.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы – создать программу на языке C, которая обрабатывает BMP-файл согласно запросам пользователя в соответствии с заявленным функционалом.

Для выполнения работы необходимо:

1. Реализовать обработку запросов пользователя в CLI
2. Создать структуры для работы с файлами
3. Реализовать считывание и запись BMP-файла
4. Организовать обработку изображения в соответствии с запросом пользователя
5. Обработка всевозможных ошибок

Для выполнения первой задачи интерфейс реализуем с помощью библиотеки getopt.h.

Для чтения и записи файлов будем использовать функции библиотеки stdio.h

Для обработки изображения будем пользоваться двумерным массивом структур-пикселей.

2. ХОД ВЫПОЛНЕНИЯ РАБОТЫ

2.1. Структуры

Так как существует выравнивание структур, в памяти могут образовываться так называемые «дырки», которые могут создать неудобства при считывании заголовков *BMP*-файла. Чтобы устранить их, воспользуемся директивами *#pragma pack(push,1)* и *#pragma pack(pop)* и обернем наши структуры в них. Первая директива устанавливает «паковку» по 1 байту, а вторая – возвращает к исходной настройке.

Далее создадим структуры *BitmapFileHeader* и *BitmapInfoHeader* с полями, которые соответствуют версии *V3* формата *BMP*.

Также создадим структуру *Rgb* с полями, соответствующими каналам RGB-компонент. Каждое поле будет принимать значение от 0 до 255, такая структура будет кодировать 1 пиксель 3 байтами, где каждый байт выделен под *RGB*-канал. А массив из таких структур будет хранить картинку.

С помощью *typedef* определим этим структурам «короткие» имена.

2.2. Интерфейс командной строки и обработка ошибок

Запрос пользователя будет представлять из себя строку:

./a.out <имя исходного файла> -<ключ1>/--<ключ1> <аргумент1>…

Для обработки запросов в функции *int main(int argc, int argv[])* будет использоваться функция *getopt()*.

Если пользователь передал программе меньше 3х аргументов, значит, точно не хватит либо имени файла, либо ключа, поэтому выводим пользователю справку о программе *printfHelp()* и завершаем программу.

Считываем из первого аргумента имя файла, которое необходимо обработать. С помощью *fopen()* открываем файл. Если файл открыть не удалось, оповещаем пользователя об ошибке открытия файла и завершаем программу.

Далее считываем из файла заголовок *BitmapFileHeader*. Если поле *signature!=0x4d42*, значит, передали не *BMP*-файл. В этом случае оповещаем пользователя и завершаем программу.

Считываем поле *headersize* заголовка *BitmapInfoHeader*. И проверяем версию *BMP*-формата. Если оно равно 40 байтам, то это версия *V3*, которая нам подходит. В других случаях версии другие, нам не подходят, оповещаем пользователя и завершаем программу.

Считываем остальные поля заголовка *BitmapInfoHeader* и проверяем глубину кодирования. Если она не равна 24, то оповещаем пользователя, что такой файл нам не подходит для обработки и завершаем программу.

Объявляем двумерный массив *Rgb* структур *Rgb** arr*, который будет хранить нашу картинку. Построчно заполняем его, считывая информацию из файла, при этом строки должны быть длины кратной 4 из-за выравнивания структур, поэтому к исходной длине строки прибавляем некоторый *padding*. Указатель на получившийся массив мы будем потом передавать во все функции, чтобы обрабатывать изображение, которое он хранит.

Далее будем обрабатывать ключи и получаемые аргументы с помощью функции *getopt()*. Обрабатывая полученные аргументы, можем выявлять ошибки по типу: введено недостаточное количество аргументов для этой функции, выбранные координаты находятся за пределами размеров картинки, значения *RGB*-компонент находятся вне диапазона 0-255 и тд. А также обрабатывая сами ключи, мы можем вызывать необходимую функцию для обработки изображения. Каждый ключ имеет краткую и длинную форму записи. Список ключей и их аргументы представлены в справке, которую пользователь может вызвать, набрав в консоли: *./a.out -h*

2.3. Инверсия цвета в заданной области

Определим функцию *void inversionColor*. Вызов функции позволяет перебрать элементы-пиксели массива-изображения в заданной области и установить в каждом таком пикселе инвертируемый цвет. Для того, чтобы

инвертировать цвет, достаточно значение каждой из *RGB*-компонент изменить на *val,* где *val* – значение *RGB*-компоненты. К каждой компоненте будем обращаться как к полю структуры *RGB.* В результате выполнения функции «начинка» массива будет изменена, передаем этот измененный массив в функцию, которая запишет его в файл вывода.

2.4. Преобразование в ЧБ заданной области

Определим функцию *void BlackWhite*. Вызов функции позволяет перебрать элементы-пиксели массива-изображения в заданной области и установить в каждом таком пикселе ЧБ-цвет. Для того, чтобы получить ЧБ версию цвета, достаточно значение каждой из *RGB*-компонент изменить на *mid,* где *mid* – среднее арифметическое всех трех *RGB*-компонент. К каждой компоненте будем обращаться как к полю структуры *RGB.* В результате выполнения функции «начинка» массива будет изменена, передаем этот измененный массив в функцию, которая запишет его в файл вывода.

2.5. Увеличение или уменьшение изображения(с обрезкой)

Определим функцию *void changeSize*. Вызов функции позволяет в зависимости от введенной пользователем операции: ‘+’ – увеличение изображения с расширением фона или ‘-‘ – уменьшение изображения с его обрезкой, в зависимости от положения, относительно которого будет выполняться действие: *leftDown,leftUp,rightDown,rightUp,center,* обрабатывает переданный массив структур. Если выбрана операция ‘+’, то если выбрано положение *leftDown,* то будет создан новый массив, в левом нижнем углу которого лежит исходный, а все остальные пиксели будут закрашены в цвет, определенный пользователем, аналогично если выбрано положение *rightUp,* то будет создан новый массив, в правом верхнем углу которого лежит исходный, а все остальные пиксели закрашены определенным цветом и т.д. Если же выбрана операция *‘-',* то будет создаваться новый массив, длина и ширина которого уменьшены с определенной стороны, нетронутые пиксели же остаются на месте.

Например, если выбрано положение *leftDown,* то от целого изображения останется только левый нижний уголок, длина уменьшится сверху на 200 пикселей, ширина уменьшится справа на 200 пикселей, то есть всё ненужное «обрежется». Аналогично для других положений. Результат вызова функции - новый массив запишется в файл *out.bmp.*

2.6. Рисование отрезка с заданными координатами

Определим функцию *void setLine.* Вызов данной функции позволяет построить отрезок указанного цвета и толщины с заданными координатами в пределах исходной картинки. Для реализации этой функции воспользуемся алгоритмом Брезенхема для целых чисел. Сначала определяем угол наклона отрезка. Для угла меньшего или равного 45 градусов реализуем алгоритм, а для угла большего 45 градусов достаточно проделать этот же алгоритм, но при этом предварительно поменять оси абсцисс и ординат местами. Если разница абсцисс точек начала и конца отрезка *dx>=* разности ординат *dy,* то угол будет меньше либо равен 45. Нам нужно рисовать отрезок во всех направлениях, вверх, вниз, вправо, влево, по диагонали вправо вниз, по диагонали влево вверх и т.д. Поэтому во время черчения отрезка *y* может изменяться, как с шагом 1, так и с 0, и с -1. Поэтому определим этот шаг в зависимости от разности *y1-y0.* Далее будем перебирать *x,* в зависимости от направления по оси абсцисс, которое можно узнать, сравнив *x1-x0?0.* Перебирая иксы в цикле *for* будем закрашивать, полученные на итерациях точки. И если толщина отрезка больше единицы, то закрашивать еще и соседние точки. Таким образом, в результате выполнения функции точки отрезка – пиксели будут окрашены в цвет, определенный пользователем. И обработанное изображение – исходное изображение с нарисованным поверх отрезком, заданным пользователем, запишется в файл *out.bmp.*

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была написана программа на языке C, имеющая CLI, для обработки изображений - BMP-файлов. Пользователь вводит имя файла, который нужно обработать, ключ функции-действия, которое нужно произвести над изображением, и аргументы к ключу, чтобы обработать изображение одним из следующих действий: инвертировать цвета в заданной области, преобразовать в ЧБ заданную область, увеличить изображение с расширением фона или уменьшить, обрезав изображение, относительно некоторого положения, нарисовать отрезок определенного цвета и толщины с заданными координатами. Полученный результат соответствует поставленной цели.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Информация о BMP-формате и его версиях. https://ru.wikipedia.org/wiki/BMP#/media/Файл:BMPfileFormat.png
2. Алгоритм Брезенхема. https://ru.wikipedia.org/wiki/Алгоритм_Брезенхэма

ПРИЛОЖЕНИЕ А
ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

```
programmer@vb:~/CLionProjects/cw2$ ./a.out -h
Прочитайте руководство по использованию программы:
        Программа обрабатывает BMP-файлы версии V3. Если подаются файлы версии CORE,V5,V4 - программа сообщает, что такая
 версия не поддерживается.
        Для запуска программы необходимо передать следующие аргументы:
        Обязательные аргументы:
                ./a.out  -- имя исполняемого файла
                <filename>  -- имя BMP-файла, который необходимо обработать. Он должен находиться в текущей директории
                -f or --func  -- ключ функции-действия
                <arg1>,<arg2> ... -- аргументы к ключу, если требуются(указано в списке ключей, их может быть несколько,
АРГУМЕНТЫ разделяются ЗАПЯТОЙ)
        Список ключей и их аргументы(если имеются):
                --info/-i  (без аргументов) -- вывод информации о BMP-файле, значения полей его заголовков
                --help/-h  (без аргументов) -- вывод руководства по использованию программы
                --inversionColor/-I <leftUpX>,<leftUpY>,<rightDownX>,<rightDownY>
                        -- инверсия цвета в заданной области, где
                                <leftUpX> - координата X левого верхнего угла области
                                <leftUpY> - координата Y левого верхнего угла области
                                <rightDownX> - координата X правого нижнего угла области
                                <rightDownY> - координата Y правого нижнего угла области
                --blackWhite/-B <leftUpX>,<leftUpY>,<rightDownX>,<rightDownY>
                        -- преобразование в ЧБ заданной области, где
                                <leftUpX> - координата X левого верхнего угла области
                                <leftUpY> - координата Y левого верхнего угла области
                                <rightDownX> - координата X правого нижнего угла области
                                <rightDownY> - координата Y правого нижнего угла области
                --blackWhite/-B <leftUpX>,<leftUpY>,<rightDownX>,<rightDownY>
                        -- преобразование в ЧБ заданной области, где
                                <leftUpX> - координата X левого верхнего угла области
                                <leftUpY> - координата Y левого верхнего угла области
                                <rightDownX> - координата X правого нижнего угла области
                                <rightDownY> - координата Y правого нижнего угла области
                --changeSize/-S <operation>,<r>,<g>,<b>,<point>
                        -- изменение размера изображения с его обрезкой или расширением фона, где
                                <operation> - символ '+', если нужно расширить фон, '-' если обрезать
                                <r> - компонента red цвета расширения фона(ввести число от 0 до 255, даже если <operation
> = '-', в таком случае оно не сыграет роли)
                                <g> - компонента green цвета расширения фона(ввести число от 0 до 255, даже если <operati
on> = '-', в таком случае оно не сыграет роли)
                                <b> - компонента blue цвета расширения фона(ввести число от 0 до 255, даже если <operatio
n> = '-', в таком случае оно не сыграет роли)
                                <point> - точка, относительно которой производится действие:
                                        на выбор: leftDown, leftUp, rightDown, rightUp, center
                --setLine/-L <x0>,<y0>,<x1>,<y1>,<r>,<g>,<b>,<thickness>
                        -- рисование отрезка, где
                                <x0> и <y0> - координаты начала отрезка
                                <x1> и <y1> - координаты конца отрезка
                                <r> - компонента red цвета отрезка
                                <g> - компонента green цвета отрезка
                                <b> - компонента blue цвета отрезка
                                <thickness> - толщина отрезка в пикселях
        Результат успешной обработки изображения записывается в файл out.bmp
programmer@vb:~/CLionProjects/cw2$
```

Пример 1. Вывод справки о программе

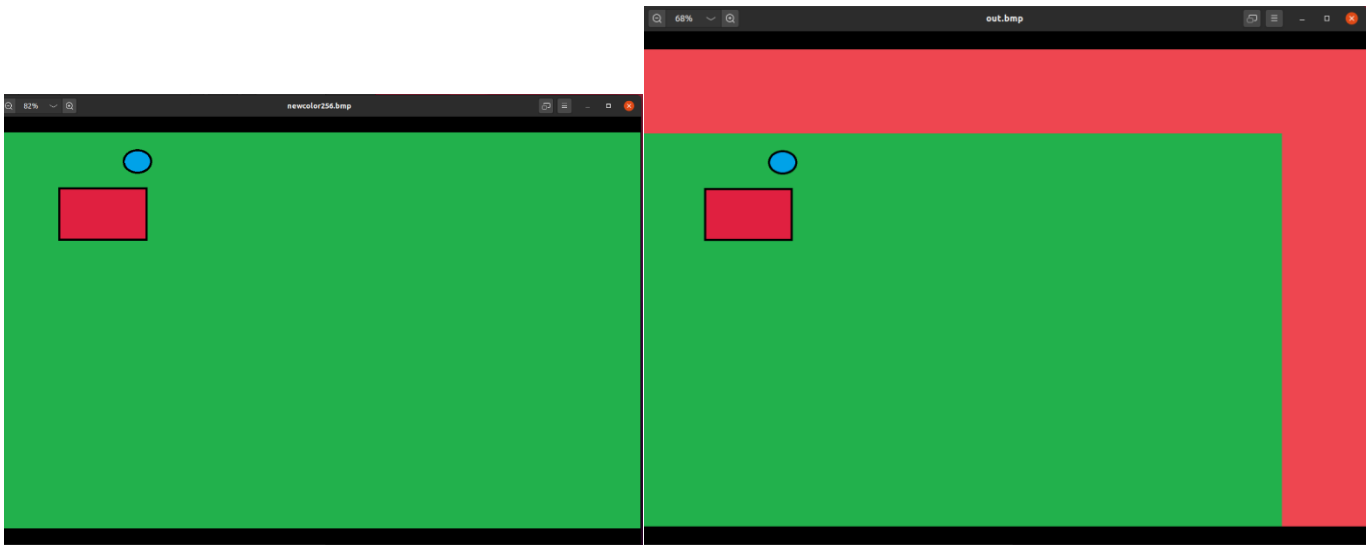
```
programmer@vb:~/CLionProjects/cw2$ ./a.out simpsonsvr.bmp -I 220,400,410,205
Результат обработки изображения помещен в файл out.bmp. Чтобы посмотреть результат, введите в консоль: eog out.bmp
programmer@vb:~/CLionProjects/cw2$ eog out.bmp
```

Пример 2. Инверсия цвета в заданной области

```
programmer@vb:~/CLionProjects/cw2$ ./a.out simpsonsvr.bmp -B 220,400,410,205
Результат обработки изображения помещен в файл out.bmp. Чтобы посмотреть результат, введите в консоль: eog out.bmp
programmer@vb:~/CLionProjects/cw2$ eog out.bmp
```

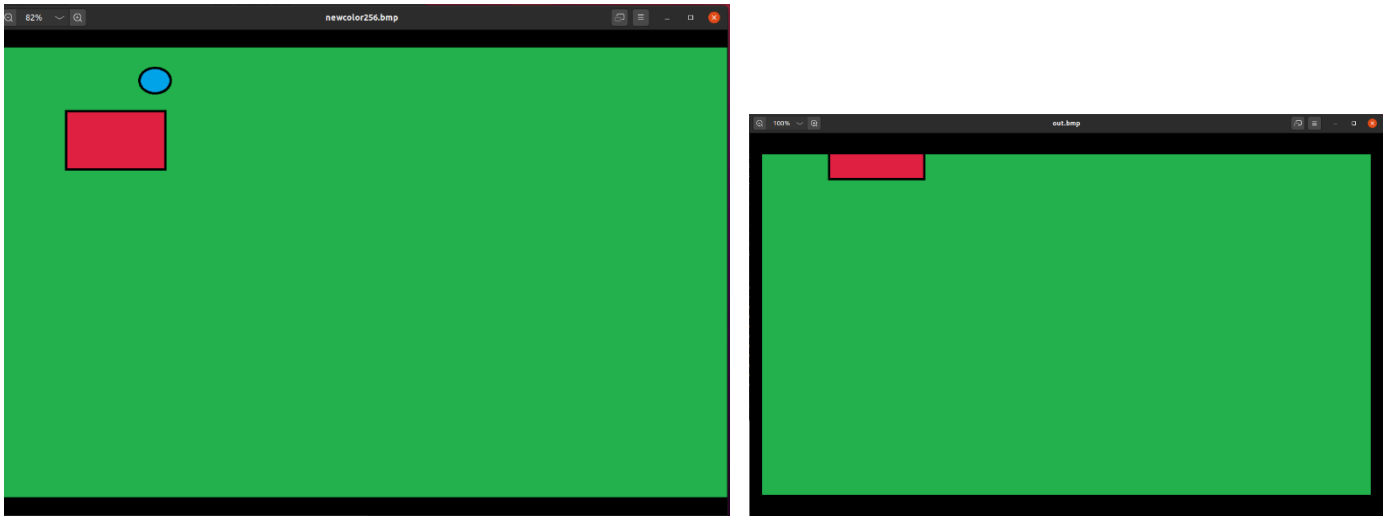
Пример 3. Преобразование в ЧБ заданной области

```
programmer@vb:~/CLionProjects/cw2$ ./a.out newcolor256.bmp -S +,238,70,80,leftDown
Результат обработки изображения помещен в файл out.bmp. Чтобы посмотреть результат, введите в консоль: eog out.bmp
programmer@vb:~/CLionProjects/cw2$ eog out.bmp
```



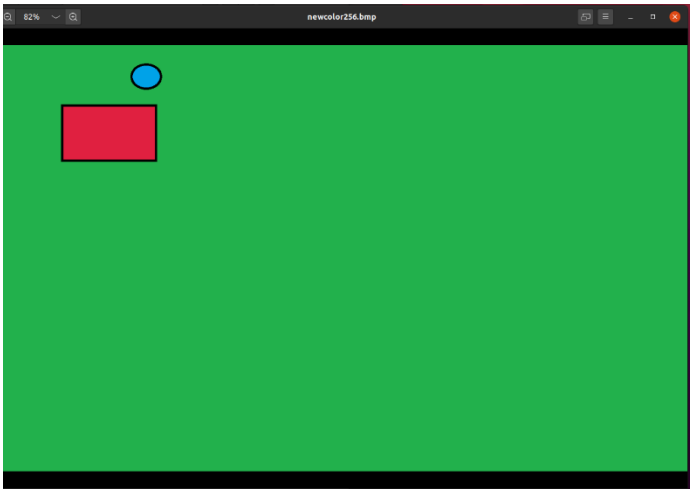
Пример 4. Изменение размера изображения. Увеличение

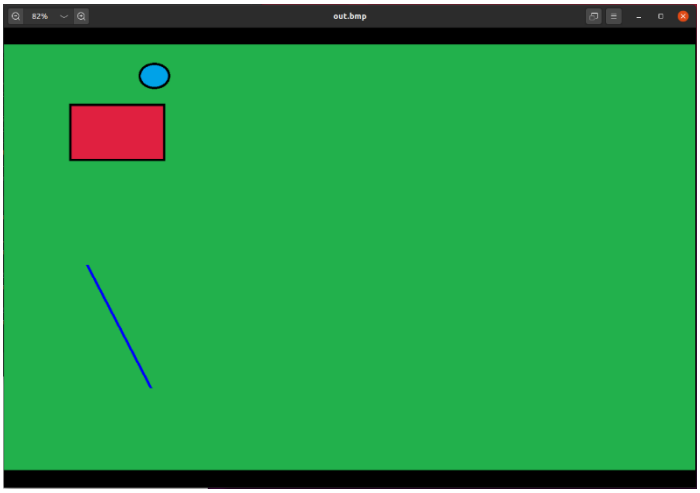
```
programmer@vb:~/CLionProjects/cw2$ ./a.out newcolor256.bmp -S -,238,70,80,leftDown
Результат обработки изображения помещен в файл out.bmp. Чтобы посмотреть результат, введите в консоль: eog out.bmp
programmer@vb:~/CLionProjects/cw2$ eog out.bmp
```



Пример 5. Изменение размера изображения. Уменьшение

```
programmer@vb:~/CLionProjects/cw2$ ./a.out newcolor256.bmp -L 180,450,320,180,0,0,255,6
Результат обработки изображения помещен в файл out.bmp. Чтобы посмотреть результат, введите в консоль: eog out.bmp
programmer@vb:~/CLionProjects/cw2$ eog out.bmp
```





Пример 6. Рисование заданного отрезка

```
programmer@vb:~/CLionProjects/cw2$ ./a.out simpsonsvr.bmp -I 1250,300,68,89
Введенные координаты должны быть в пределах размера картинки
programmer@vb:~/CLionProjects/cw2$ ./a.out simpsonsvr.bmp -I 100,20,180,40
Из таких координат прямоугольной области не выходит. Проверьте, (x0,y0) - координаты ЛЕВОГО ВЕРХНЕГО угла, (x1,н1) - коор
динаты ПРАВОГО НИЖНЕГО угла
programmer@vb:~/CLionProjects/cw2$ ./a.out simpsonsvr.bmp -B 20,10,30
Для преобразования в ЧБ некорректно введены аргументы
Прочитайте руководство по использованию программы:
```

```
programmer@vb:~/CLionProjects/cw2$ ./a.out simpsonsvr.bmp -S *,200,300,400,100,leftDown
Аргумент <operation> может быть либо '+', либо '-', подробнее смотри в справке(./a.out -h)
```

```
programmer@vb:~/CLionProjects/cw2$ ./a.out simpsonsvr.bmp -S -,200,300,400,100,superRight
Аргумент <point> может быть одним из 5 значений: leftDown,leftUp,rightDown,rightUp,center,
 подробнее смотри в справке(./a.out -h)
```

```
programmer@vb:~/CLionProjects/cw2$ ./a.out simpsonsvr.bmp -L 200,1240,300,290,0,0,0,5
Координаты отрезка должны быть в пределах размеров картинки, RGB-компоненты в диапазоне от 0 до 255, толщина отрезка больше 0
```

Пример 7. Вывод ошибок

ПРИЛОЖЕНИЕ В
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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

#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 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;
} BitmapInfoHeader;

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

#pragma pack(pop)

void othersRead(FILE* f,BitmapInfoHeader* bmih){ //функция
считывания остальных полей стр-ры BitmapInfoHeader,
                                                  // помимо
headerSize
    fread(&bmih->width, 1, sizeof(bmih->width), f);
    fread(&bmih->height, 1, sizeof(bmih->height), f);
    fread(&bmih->planes, 1, sizeof(bmih->planes), f);
    fread(&bmih->bitsPerPixel, 1, sizeof(bmih->bitsPerPixel), f);
    fread(&bmih->compression, 1, sizeof(bmih->compression), f);
    fread(&bmih->imageSize, 1, sizeof(bmih->imageSize), f);
    fread(&bmih->xPixelsPerMeter, 1,
sizeof(bmih->xPixelsPerMeter), f);
```

```
          fread(&bmih->yPixelsPerMeter, 1,
sizeof(bmih->yPixelsPerMeter), f);
          fread(&bmih->colorsInColorTable, 1,
sizeof(bmih->colorsInColorTable), f);
          fread(&bmih->importantColorCount, 1,
sizeof(bmih->importantColorCount), f);
      }

      void printFileInfoHeader(BitmapFileHeader bmfh,BitmapInfoHeader
bmih){
          printf("Информация о BMP-файле:\n");
          printf("\tПоля структуры BitmapFileHeader:\n");
          printf("\t\tsignature:\t%x (%hu)\n", bmfh.signature,
bmfh.signature);
          printf("\t\tfilesize:\t%x (%u)\n", bmfh.filesize,
bmfh.filesize);
          printf("\t\treserved1:\t%x (%hu)\n", bmfh.reserved1,
bmfh.reserved1);
          printf("\t\treserved2:\t%x (%hu)\n", bmfh.reserved2,
bmfh.reserved2);
          printf("\t\tpixelArrOffset:\t%x (%u)\n", bmfh.pixelArrOffset,
bmfh.pixelArrOffset);

          printf("\tПоля структуры BitmapInfoHeader:\n");
          printf("\t\theaderSize:\t%x (%u)\n", bmih.headerSize,
bmih.headerSize);
          printf("\t\twidth:     \t%x (%u)\n", bmih.width, bmih.width);
          printf("\t\theight:    \t%x (%u)\n", bmih.height,
bmih.height);
          printf("\t\tplanes:    \t%x (%hu)\n", bmih.planes,
bmih.planes);
          printf("\t\tbitsPerPixel:\t%x (%hu)\n", bmih.bitsPerPixel,
bmih.bitsPerPixel);
          printf("\t\tcompression:\t%x (%u)\n", bmih.compression,
bmih.compression);
          printf("\t\timageSize:\t%x (%u)\n", bmih.imageSize,
bmih.imageSize);
          printf("\t\txPixelsPerMeter:\t%x (%u)\n",
bmih.xPixelsPerMeter, bmih.xPixelsPerMeter);
          printf("\t\tyPixelsPerMeter:\t%x (%u)\n",
bmih.yPixelsPerMeter, bmih.yPixelsPerMeter);
          printf("\t\tcolorsInColorTable:\t%x (%u)\n",
bmih.colorsInColorTable, bmih.colorsInColorTable);
          printf("\t\timportantColorCount:\t%x (%u)\n",
bmih.importantColorCount, bmih.importantColorCount);
      }
      void extendWriting(Rgb** arr,unsigned int startH,unsigned int
endH,unsigned int startW,unsigned int endW, BitmapFileHeader* bmfh,
BitmapInfoHeader* bmih){
          FILE* ff=fopen("out.bmp","wb");



          fclose(ff);
      }
```

```
     void defaultWriting(Rgb** arr, unsigned int startH, unsigned int
endH, unsigned int startW, unsigned endW, BitmapFileHeader* bmfh,
BitmapInfoHeader* bmih){
          FILE *ff = fopen("out.bmp", "wb");      //create a file-
descriptor for a file with a result-picture

          bmih->height = endH-startH;
          bmih->width = endW-startW;
          fwrite(bmfh, 1, sizeof(BitmapFileHeader),ff);
          fwrite(bmih, 1, sizeof(BitmapInfoHeader),ff);
          unsigned int w = (endW-startW) * sizeof(Rgb) + ((4-((endW-
startW)*sizeof(Rgb)%4))&3);
          Rgb* str=calloc(w,sizeof(Rgb));
          for(unsigned int i=startH; i<endH; i++) {
              int ind = 0;
              for (unsigned int j = startW; j < endW; j++) {
                  str[ind++] = arr[i][j];
              }
              fwrite(str, 1, w, ff);
          }
          fclose(ff);
     }

     void inversionColor(Rgb** arr,int leftUpX,int leftUpY, int
rightDownX, int rightDownY, BitmapFileHeader* bmfh, BitmapInfoHeader*
bmih){
          for(int i=rightDownY;i<leftUpY;i++)
              for(int j=leftUpX;j<rightDownX;j++){
                  arr[i][j].b=255-arr[i][j].b;
                  arr[i][j].g=255-arr[i][j].g;
                  arr[i][j].r=255-arr[i][j].r;
              }
          defaultWriting(arr,0,bmih->height,0,bmih->width,bmfh,bmih);
     }

     void blackWhite(Rgb** arr,int leftUpX,int leftUpY, int
rightDownX, int rightDownY, BitmapFileHeader* bmfh, BitmapInfoHeader*
bmih){
          for(int i=rightDownY;i<leftUpY;i++)
              for(int j=leftUpX;j<rightDownX;j++){
                  unsigned char
mid=(arr[i][j].b+arr[i][j].g+arr[i][j].g)/3;
                  arr[i][j].b=mid;
                  arr[i][j].g=mid;
                  arr[i][j].r=mid;
              }
          defaultWriting(arr,0,bmih->height,0,bmih->width,bmfh,bmih);
     }

     void createArr(Rgb** arr, int r, int g, int b, unsigned int
startY, unsigned int endY, unsigned int startX, unsigned int endX,
unsigned int endH, unsigned int endW,
                    BitmapFileHeader* bmfh, BitmapInfoHeader* bmih){
          Rgb** newArr=malloc(endH*sizeof(Rgb*));
          unsigned int indY=0;
          for(int i=0;i<endH;i++){
```

```
            unsigned int w=endW*sizeof(Rgb)+((4-
(endW*sizeof(Rgb)%4))&3);
            newArr[i]=malloc(w*sizeof(Rgb));
            unsigned int indX=0, flag=0;
            for(int j=0;j<endW;j++){
                if(i>=startY&&i<endY&&j>=startX&&j<endX){
                    newArr[i][j]=arr[indY][indX++];
                    flag=1;
                }
                else{
                    newArr[i][j].r=r;
                    newArr[i][j].g=g;
                    newArr[i][j].b=b;
                }
            }
            if(flag==1)
                indY++;
        }
        defaultWriting(newArr,0,endH,0,endW,bmfh,bmih);

    }

    void changeSize(char c, char* startPoint, int r, int g, int b,
Rgb** arr, BitmapFileHeader* bmfh, BitmapInfoHeader* bmih){
        unsigned int offset=200;
        if(c=='+'){
            if(strcmp(startPoint,"leftDown")==0){
                unsigned int startY=0;
                unsigned int endY=bmih->height;
                unsigned int startX=0;
                unsigned int endX=bmih->width;
                unsigned int endH=offset+bmih->height;
                unsigned int endW=offset+bmih->width;

createArr(arr,r,g,b,startY,endY,startX,endX,endH,endW,bmfh,bmih);
            }
            if(strcmp(startPoint,"leftUp")==0){
                unsigned int startY=offset;
                unsigned int endY=offset+bmih->height;
                unsigned int startX=0;
                unsigned int endX=bmih->width;
                unsigned int endH=offset+bmih->height;
                unsigned int endW=offset+bmih->width;

createArr(arr,r,g,b,startY,endY,startX,endX,endH,endW,bmfh,bmih);
            }
            if(strcmp(startPoint,"rightDown")==0){ //something wrong
== leftDown
                unsigned int startY=0;
                unsigned int endY=bmih->height;
                unsigned int startX=offset;
                unsigned int endX=offset+bmih->width;
                unsigned int endH=offset+bmih->height;
                unsigned int endW=offset+bmih->width;

createArr(arr,r,g,b,startY,endY,startX,endX,endH,endW,bmfh,bmih);
            }
```

```
            if(strcmp(startPoint,"rightUp")==0){
                unsigned int startY=offset;
                unsigned int endY=offset+bmih->height;
                unsigned int startX=offset;
                unsigned int endX=offset+bmih->width;
                unsigned int endH=offset+bmih->height;
                unsigned int endW=offset+bmih->width;

createArr(arr,r,g,b,startY,endY,startX,endX,endH,endW,bmfh,bmih);
            }
            if(strcmp(startPoint,"center")==0){
                unsigned int startY=offset;
                unsigned int endY=offset+bmih->height;
                unsigned int startX=offset;
                unsigned int endX=offset+bmih->width;
                unsigned int endH=2*offset+bmih->height;
                unsigned int endW=2*offset+bmih->width;

createArr(arr,r,g,b,startY,endY,startX,endX,endH,endW,bmfh,bmih);
            }
            return;
        }
        if(c=='-'){
            if(strcmp(startPoint,"leftDown")==0){
                unsigned int newH=bmih->height-offset;
                unsigned int newW=bmih->width-offset;
                defaultWriting(arr,0,newH,0,newW,bmfh,bmih);
            }
            if(strcmp(startPoint,"leftUp")==0){
                unsigned int startH=offset;
                unsigned int endW=bmih->width-offset;

defaultWriting(arr,startH,bmih->height,0,endW,bmfh,bmih);
            }
            if(strcmp(startPoint,"rightDown")==0){ //something wrong
== leftDown
                unsigned int endH=bmih->height-offset;
                unsigned int startW=offset;

defaultWriting(arr,0,endH,startW,bmih->width,bmfh,bmih);
            }

if(strcmp(startPoint,"rightUp")==0){            //something wrong ==
leftUp
                unsigned int startH=offset;
                unsigned int startW=offset;

defaultWriting(arr,startH,bmih->height,startW,bmih->width,bmfh,bmih);
            }

if(strcmp(startPoint,"center")==0){            //something wrong ==
leftUp
                unsigned int startH=offset/2;
                unsigned int endH=bmih->height-(offset/2);
                unsigned int startW=offset/2;
                unsigned int endW=bmih->width-(offset/2);
```

```
defaultWriting(arr,startH,endH,startW,endW,bmfh,bmih);
            }
            return;
        }
        printf("Wrong operation to change size of picture. Try use
'+' - to get a bigger size, '-' - to get a smaller size\n");
    }

    void setPoint(Rgb** arr, int x, int y, unsigned int r, unsigned
int g, unsigned int b){
        arr[y][x].r=r;
        arr[y][x].g=g;
        arr[y][x].b=b;
    }

    void setLine(Rgb** arr,int x0,int y0,int x1,int y1,int r, int g,
int b,int thickness,BitmapFileHeader* bmfh,BitmapInfoHeader* bmih){
        int absDeltaX=abs(x1-x0);
        int absDeltaY=abs(y1-y0);

        int growth=0;

        if(absDeltaX>=absDeltaY){    //if angle Fi<=45
            int y=y0;
            int direction;
            if(y1-y0>0)
                direction=1;
            if(y1-y0==0)
                direction=0;
            if(y1-y0<0)
                direction=-1;
            if(x1-x0>0){
                for(int x=x0;x<=x1;x++){
                    setPoint(arr,x,y,r,g,b);
                    if(thickness!=1){
                        if(y0==y1){
                            for(int i=1;i<thickness;i++){
                                setPoint(arr,x,y+i,r,g,b);
                            }
                        }
                        else
                            for(int i=1;i<thickness;i++){
                            setPoint(arr,x+i,y,r,g,b);
                            }
                    }
                    growth+=absDeltaY;
                    if(growth>=absDeltaX){
                        growth-=absDeltaX;
                        y+=direction;
                    }
                }
            }
            if(x1-x0<0){
                for(int x=x0;x>=x1;x--){
                    setPoint(arr,x,y,r,g,b);
                    if(thickness!=1){
```

```
                    if(y0==y1){
                        for(int i=1;i<thickness;i++){
                            setPoint(arr,x,y+i,r,g,b);
                        }
                    }
                    else
                        for(int i=1;i<thickness;i++){
                            setPoint(arr,x+i,y,r,g,b);
                        }
                }
                growth+=absDeltaY;
                if(growth>=absDeltaX){
                    growth-=absDeltaX;
                    y+=direction;
                }
            }
        }

    }
    else{          //change the axis of X and Y
        int x=x0;
        int direction;
        if(x1-x0>0)
            direction=1;
        if(x1-x0==0)
            direction=0;
        if(x1-x0<0)
            direction=-1;
        if(y1-y0>0){
            for(int y=y0;y<=y1;y++){
                setPoint(arr,x,y,r,g,b);
                if(thickness!=1){
                    if(y0==y1){
                        for(int i=1;i<thickness;i++){
                            setPoint(arr,x,y+i,r,g,b);
                        }
                    }
                    else
                        for(int i=1;i<thickness;i++){
                            setPoint(arr,x+i,y,r,g,b);
                        }
                }
                growth+=absDeltaX;
                if(growth>=absDeltaY){
                    growth-=absDeltaY;
                    x+=direction;
                }
            }
        }
        if(y1-y0<0){
            for(int y=y0;y>=y1;y--){
                setPoint(arr,x,y,r,g,b);
                if(thickness!=1){
                    if(y0==y1){
                        for(int i=1;i<thickness;i++){
                            setPoint(arr,x,y+i,r,g,b);
                        }
```

```
                        }
                        else
                            for(int i=1;i<thickness;i++){
                                setPoint(arr,x+i,y,r,g,b);
                            }
                    }
                    growth+=absDeltaX;
                    if(growth>=absDeltaY){
                        growth-=absDeltaY;
                        x+=direction;
                    }
                }
            }
        }
        defaultWriting(arr,0,bmih->height,0,bmih->width,bmfh,bmih);
    }
    void printHelp(){
        printf("Прочитайте руководство по использованию
программы:\n");
        printf("\tПрограмма обрабатывает BMP-файлы версии V3. Если
подаются файлы версии CORE,V5,V4 - программа сообщает, что такая
версия не поддерживается.\n");
        printf("\tДля запуска программы необходимо передать следующие
аргументы:\n");
        printf("\tОбязательные аргументы:\n");
        printf("\t\t./a.out  -- имя исполняемого файла\n");
        printf("\t\t<filename>  -- имя BMP-файла, который необходимо
обработать. Он должен находиться в текущей директории\n");
        printf("\t\t-f or --func  -- ключ функции-действия\n");
        printf("\t\t<arg1>,<arg2> ... -- аргументы к ключу, если
требуются(указано в списке ключей, их может быть несколько, АРГУМЕНТЫ
разделяются ЗАПЯТОЙ)\n");
        printf("\tСписок ключей и их аргументы(если имеются):\n");
        printf("\t\t--info/-i  (без аргументов) -- вывод информации о
BMP-файле, значения полей его заголовков\n");
        printf("\t\t--help/-h  (без аргументов) -- вывод руководства
по использованию программы\n");
        printf("\t\t--inversionColor/-I
<leftUpX>,<leftUpY>,<rightDownX>,<rightDownY>\n");
        printf("\t\t\t-- инверсия цвета в заданной области, где\n");
        printf("\t\t\t\t<leftUpX> - координата X левого верхнего угла
области\n");
        printf("\t\t\t\t<leftUpY> - координата Y левого верхнего угла
области\n");
        printf("\t\t\t\t<rightDownX> - координата X правого нижнего
угла области\n");
        printf("\t\t\t\t<rightDownY> - координата Y правого нижнего
угла области\n");
        printf("\t\t--blackWhite/-B
<leftUpX>,<leftUpY>,<rightDownX>,<rightDownY>\n");
        printf("\t\t\t-- преобразование в ЧБ заданной области,
где\n");
        printf("\t\t\t\t<leftUpX> - координата X левого верхнего угла
области\n");
        printf("\t\t\t\t<leftUpY> - координата Y левого верхнего угла
области\n");
```

```
        printf("\t\t\t\t<rightDownX> - координата X правого нижнего
угла области\n");
        printf("\t\t\t\t<rightDownY> - координата Y правого нижнего
угла области\n");
        printf("\t\t--changeSize/-S
<operation>,<r>,<g>,<b>,<point>\n");
        printf("\t\t\t-- изменение размера изображения с его обрезкой
или расширением фона, где\n");
        printf("\t\t\t\t<operation> - символ '+', если нужно
расширить фон, '-' если обрезать\n");
        printf("\t\t\t\t<r> - компонента red цвета расширения
фона(ввести число от 0 до 255, даже если <operation> = '-', в таком
случае оно не сыграет роли)\n");
        printf("\t\t\t\t<g> - компонента green цвета расширения
фона(ввести число от 0 до 255, даже если <operation> = '-', в таком
случае оно не сыграет роли)\n");
        printf("\t\t\t\t<b> - компонента blue цвета расширения
фона(ввести число от 0 до 255, даже если <operation> = '-', в таком
случае оно не сыграет роли)\n");
        printf("\t\t\t\t<point> - точка, относительно которой
производится действие:\n");
        printf("\t\t\t\t\tна выбор: leftDown, leftUp, rightDown,
rightUp, center\n");
        printf("\t\t--setLine/-L
<x0>,<y0>,<x1>,<y1>,<r>,<g>,<b>,<thickness>\n");
        printf("\t\t\t-- рисование отрезка, где\n");
        printf("\t\t\t\t<x0> и <y0> - координаты начала отрезка\n");
        printf("\t\t\t\t<x1> и <y1> - координаты конца отрезка\n");
        printf("\t\t\t\t<r> - компонента red цвета отрезка\n");
        printf("\t\t\t\t<g> - компонента green цвета отрезка\n");
        printf("\t\t\t\t<b> - компонента blue цвета отрезка\n");
        printf("\t\t\t\t<thickness> - толщина отрезка в пикселях\n");
        printf("\tРезультат успешной обработки изображения
записывается в файл out.bmp\n");
    }

    int main(int argc, char* argv[]) {
        setlocale(LC_ALL,"");

        int leftUpX=-1,leftUpY=-1,rightDownX=-1,rightDownY=-1;
        int r=-1,g=-1,b=-1;
        char c='\0';
        char* startPoint=calloc(100,sizeof(char));
        int x0=-1,y0=-1,x1=-1,y1=-1,thickness=-1;
        int flag=0;

        if(argc<3){
            printHelp();
            return 0;
        }

        char* fname=calloc(100,sizeof(char));
        strcpy(fname,argv[1]);

        FILE *f = fopen(fname, "rb");
        if(!f){
```

```
            printf("Такого файла не существует в текущей
директории.\n");
            return 0;
        }
        BitmapFileHeader bmfh;
        BitmapInfoHeader bmih;
        fread(&bmfh,1,sizeof(BitmapFileHeader),f);

        if(bmfh.signature!=0x4d42){          //checking BMP-format
            printf("It's not BMP-format\n");
            return 0;
        }
        fread(&bmih.headerSize,1,sizeof(bmih.headerSize),f);
//checking version of BMP-format
        switch (bmih.headerSize)
{                                  //excluding CORE,V4,V5 versions
            case 12:
                printf("CORE version of BMP-format is not
supported\n");
                return 0;
            case 108:
                printf("V4 version of BMP-format is not
supported\n");
                return 0;
            case 124:
                printf("V5 version of BMP-format is not
supported\n");
                return 0;
        }

        othersRead(f,&bmih);                //reading others fields
        if(bmih.bitsPerPixel!=24){
            printf("Picture with bitsPerPixel!=24 is not
supported\n");
            return 0;
        }

        //reading PixelArray

        unsigned int H = bmih.height;       //create new variables
for faster code writing
        unsigned int W = bmih.width;

        Rgb **arr = malloc(H*sizeof(Rgb*));  //create an array of
pointers to strings that consist of RGB-structures(pixels)
        for(int i=0; i<H; i++){
            arr[i] = malloc(W * sizeof(Rgb) + ((4-
((W)*sizeof(Rgb)%4))&3));
            fread(arr[i],1,W * sizeof(Rgb) + ((4-
((W)*sizeof(Rgb)%4))&3),f);
        }

        char *opts = "hiI:B:S:L:";
        struct option longOpts[]={
                {"help",no_argument,NULL,'h'},
                {"inversionColor", required_argument, NULL, 'I'},
                {"blackWhite", required_argument, NULL, 'B'},
```

```
                {"changeSize", required_argument, NULL, 'S'},
                {"setLine", required_argument, NULL, 'L'},
                {"info",no_argument,NULL,'i'},
                { NULL, 0, NULL, 0}
        };
        int opt;
        int longIndex;
        opt = getopt_long(argc, argv, opts, longOpts, &longIndex);
        while(opt!=-1){
            switch(opt){
                case 'h':
                    printHelp();
                    break;
                case 'I':
                    sscanf(optarg,"%d,%d,%d,%d\n",
&leftUpX,&leftUpY,&rightDownX,&rightDownY);
                    if(leftUpX!=-1&&leftUpY!=-1&&rightDownX!=-
1&&rightDownY!=-1){

if(leftUpX<0||rightDownX<0||leftUpY<0||rightDownY<0||leftUpX>=bmih.wid
th||rightDownX>=bmih.width||leftUpY>=bmih.height||rightDownY>=bmih.hei
ght){
                            printf("Введенные координаты должны быть
в пределах размера картинки\n");
                            return 0;
                        }
                        if(leftUpY<rightDownY||rightDownX<leftUpX){
                            printf("Из таких координат прямоугольной
области не выходит. Проверьте, (x0,y0) - координаты ЛЕВОГО ВЕРХНЕГО
угла, (x1,н1) - координаты ПРАВОГО НИЖНЕГО угла\n");
                            return 0;
                        }

inversionColor(arr,leftUpX,leftUpY,rightDownX,rightDownY,&bmfh,&bmih);
                        flag=1;
                    }
                    else{
                        printf("Для инверсии цвета некорректно
введены аргументы\n");
                        printHelp();
                        return 0;
                    }
                    break;
                case 'B':
                    sscanf(optarg,"%d,%d,%d,%d\n",
&leftUpX,&leftUpY,&rightDownX,&rightDownY);
                    if(leftUpX!=-1&&leftUpY!=-1&&rightDownX!=-
1&&rightDownY!=-1){

if(leftUpX<0||rightDownX<0||leftUpY<0||rightDownY<0||leftUpX>=bmih.wid
th||rightDownX>=bmih.width||leftUpY>=bmih.height||rightDownY>=bmih.hei
ght){
                            printf("Введенные координаты должны быть
в пределах размера картинки\n");
                            return 0;
                        }
                        if(leftUpY<rightDownY||rightDownX<leftUpX){
```

```
                                printf("Из таких координат прямоугольной
области не выходит. Проверьте, (x0,y0) - координаты ЛЕВОГО ВЕРХНЕГО
угла, (x1,н1) - координаты ПРАВОГО НИЖНЕГО угла\n");
                                return 0;
                            }

blackWhite(arr,leftUpX,leftUpY,rightDownX,rightDownY,&bmfh,&bmih);
                            flag=1;
                        }
                        else{
                            printf("Для преобразования в ЧБ некорректно
введены аргументы\n");
                            printHelp();
                            return 0;
                        }
                        break;
                    case 'S':

sscanf(optarg,"%c,%d,%d,%d,%s",&c,&r,&g,&b,startPoint);
                        if(c!='+'&&c!='-'){
                            printf("Аргумент <operation> может быть либо
'+', либо '-', подробнее смотри в справке(./a.out -h) \n");
                            return 0;
                        }

if(strcmp(startPoint,"leftDown")!=0&&strcmp(startPoint,"leftUp")!=0&&s
trcmp(startPoint,"rightDown")!=0&&strcmp(startPoint,"rightUp")!=0&&str
cmp(startPoint,"center")){
                            printf("Аргумент <point> может быть одним из
5 значений: leftDown,leftUp,rightDown,rightUp,center,\n подробнее
смотри в справке(./a.out -h) \n");
                            return 0;
                        }
                        if(c=='-'){
                            r=0;
                            g=0;
                            b=0;

changeSize(c,startPoint,r,g,b,arr,&bmfh,&bmih);
                            flag=1;
                        }
                        if(c=='+') {
                            if (r != -1 && g != -1 && b != -1) {
                                if (r >= 0 && r <= 255 && g >= 0 && g <=
255 && b >= 0 && b < 255) {
                                    changeSize(c, startPoint, r, g, b,
arr, &bmfh, &bmih);
                                    flag=1;
                                } else {
                                    printf("Компоненты RGB должны
находиться в диапазоне от 0 до 255\n");
                                    return 0;
                                }
                            } else {
                                printf("Для обрезки или расширения фона
некорректно введены аргументы\n");
                                printHelp();
```

```
                                return 0;
                            }
                        }
                        break;
                    case 'L':

sscanf(optarg,"%d,%d,%d,%d,%d,%d,%d,%d",&x0,&y0,&x1,&y1,&r,&g,&b,&thic
kness);
                        if(x0!=-1&&y0!=-1&&x1!=-1&&y1!=-1&&r!=-1&&g!=-
1&&b!=-1&&thickness!=-1){

if(x0>=0&&y0>=0&&x1>=0&&y1>=0&&r>=0&&g>=0&&b>=0&&x0<bmih.width&&x1<bmi
h.width&&y0<bmih.height&&y1<bmih.height&&r<256&g<256&&b<256&&thickness
>0){

setLine(arr,x0,y0,x1,y1,r,g,b,thickness,&bmfh,&bmih);
                                flag=1;
                            }
                            else{
                                printf("Координаты отрезка должны быть в
пределах размеров картинки, RGB-компоненты в диапазоне от 0 до 255,
толщина отрезка больше 0\n");
                                return 0;
                            }
                        }
                        else{
                            printf("Для рисования отрезка некорректно
введены аргументы\n");
                            printHelp();
                            return 0;
                        }
                        break;
                    case 'i':
                        printFileInfoHeader(bmfh,bmih);
                        break;
                    case '?':
                        printHelp();
                        return 0;
                }
                opt = getopt_long(argc, argv, opts, longOpts,
&longIndex);
            }
            argc -= optind;
            argv += optind;
            fclose(f);
            if(flag==1)
                printf("Результат обработки изображения помещен в файл
out.bmp. Чтобы посмотреть результат, введите в консоль: eog
out.bmp\n");
            return 0;
        }
```