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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка изображений в формате BMP на языке Си.

Студент гр. 1382 _____ Тапеха В.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2022

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Тапеха В.А.

Группа 1382

Тема работы: Обработка изображений в формате BMP на языке Си.

Исходные данные:
Программа принимает на вход изображение в формате BMP. Его необходимо изменить в соответствии с условиями. Также программа принимает аргументы, которые описывают, каким образом должно измениться изображение. Весь функционал реализован с использованием CLI.

Содержание пояснительной записки:
«Содержание», «Введение», «Задание работы», «Ход выполнения работы», «Тестирование», «Заключение», «Список использованных источников», «Приложение А. Исходный код программы»

Предполагаемый объем пояснительной записки:
Не менее 15 страниц.

Дата выдачи задания: 22.03.2022

Дата сдачи реферата:

Дата защиты реферата:

Студент _____ Тапеха В.А.

Преподаватель _____ Жангиров Т.Р.

АННОТАЦИЯ

В ходе выполнения курсовой работы создавалась программа на языке C для обработки изображений в формате BMP в соответствии с заданными условиями. Программа имеет CLI (Command Line Interface) с возможностью вывода инструкции о программе, которая содержит краткую справку о всех возможных командах и ключах программы. Программа поддерживает только файлы BMP version 3, без сжатия, не содержащие таблицу цветов и имеющие 24 бита на цвет. При вводе неправильных данных пользователю выводятся сообщения о соответствующих ошибках.

SUMMARY

During the course work, a C program was created for processing images in BMP format in accordance with the specified conditions. The program has a CLI (Command Line Interface) with the ability to display instructions about the program, which contains brief information about all possible commands and program keys. The program only supports BMP version 3 files, without compression, without a color table and with 24 bits per color. If the data is entered incorrectly, the user will receive messages about the corresponding errors.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы: создание программы для редактирования изображений в формате BMP на языке C с CLI, обрабатывающая изображение согласно условиям, которые описаны в задании.

Для выполнения необходимо:

1. Реализовать обработку запросов пользователя.
2. Создание структур для работы с файлами.
3. Реализация считывания и записи изображения в формате BMP.
4. Создание всех функций, описанных в задании.
5. Обработка возможных ошибок.

Для первой задачи необходимо подключить заголовочной файл getopt.h и использовать его.

Для реализации других задач достаточно использования функций, которые подключаются с помощью заголовочных файлов stdlib.h, string.h и stdio.h .

1. ЗАДАНИЕ РАБОТЫ

Вариант 10

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. Отражение заданной области. Этот функционал определяется:
    - выбором оси относительно которой отражать (горизонтальная или вертикальная)
    - Координатами левого верхнего угла области
    - Координатами правого нижнего угла области
2. Копирование заданной области. Функционал определяется:
    - Координатами левого верхнего угла области-источника
    - Координатами правого нижнего угла области-источника
    - Координатами левого верхнего угла области-назначения
3. Заменяет все пиксели одного заданного цвета на другой цвет. Функционал определяется:
    - Цвет, который требуется заменить
    - Цвет на который требуется заменить

4. Фильтр rgb-компонент. Этот инструмент должен позволять для всего изображения либо установить в 0 либо установить в 255 значение заданной компоненты. Функционал определяется
    - Какую компоненту требуется изменить
    - В какой значение ее требуется изменить

2. ХОД ВЫПОЛНЕНИЯ РАБОТЫ

2.1. Структуры

Для начала обращаем внимание на выравнивание структур. Чтобы не возникло проблем при работе с BMP-файлами оборачиваем структуры в #pragma pack(push, 1) и #pragma pack(pop). Это помогает установить размер выравнивания в 1 байт и вернуться к стандартному выравниваю. Это необходимо, чтобы не возникало пустых пространств в памяти, так как в противном случае могут возникнуть проблем при считывании изображений формата BMP.

Между #pragma pack(push, 1) и #pragma pack(pop) создаются структуры BitmapFileHeader, BitmapInfoHeader и RGB. BitmapFileHeader, BitmapInfoHeader и двумерный массив из структур RGB объединены в одну структуру BMP для удобства работы с ними. Первые две структуры содержат в себе поля необходимые для правильной работы BMP-файлов третьей версии. Структура RGB содержит информацию об одном пикселе — там хранятся поля r, g, b которые отвечают за значения красной, зеленой и синей компонент для определенного пикселя.

2.2. Считывание изображения

Чтение изображения происходит с помощью функции *int readImage(BMP *image, char* path)*. Сначала идет проверка названия файла. Необходимо, чтобы последние 4 символа всегда заканчивались на «.bmp». Затем с помощью функции fopen() открывается файл на чтение. После при помощи функции fread считывается считывается часть файла и записывается в объединенную структуру BMP в то поле, которое отвечает за BitmapFileHeader. Потом проверяется сигнатура файла, которая должна соответствовать «0x4d42». Затем при помощи функции fread считывается считывается часть файла и записывается в объединенную структуру BMP в то поле, которое отвечает за BitmapInfoHeader. После идет проверка с помощью функции *int IsImageOk(BMP image)*. Она проверяет версию BMP-файла, сжато ли изображение, используется

ли цветовая таблица, глубину цвета и количество планок. После проверки идет считывание файла по пикселям с учетом сдвига. Если памяти недостаточно, то пользователь получает сообщение об этом. После считывания файл закрывается с помощью функции fclose().

2.3. Запись изображения

В самом начале проверяется имя файла, в который должно быть записано изображение. Необходимо, чтобы название заканчивалось на «.bmp». После с помощью функции fopen() открывается по указанному пути файл. Затем с помощью fwrite() записываются поля header и info, которые отвечают за BitmapFileHeader, BitmapInfoHeader. После с помощью этой же функции записывается двумерный массив пикселей изображения. В конце с помощью функции fclose() закрывается открытый файл и чистится память с помощью функции freeMem(), созданной раннее.

2.4. Первая функция

Первая функция *reflectArea(BMP* image, char* axis, int x_left, int y_top, int x_right, int y_bottom).* Она отражает некоторую область изображения по горизонтальной или вертикальной оси. Сначала идет проверка введенных координат. Если координаты неправильные, то печатается соответствующее сообщение и функция завершает свою работу. Затем вычисляется длина и ширина выделенной области. Потом редактируются введенные координаты для того, чтобы пользователю было удобнее работать с изображением. Это нужно, потому что первый пиксель в BMP-файлах находится в левом нижнем углу, а не в левом верхнем. Дальше, исходя из того, по какой оси пользователь хочет отразить область, программа меняет пиксели местами. Если по горизонтальной оси, то пиксели по оси y остаются нетронутыми, а по оси x пиксели меняются местами относительно середины изображения. Если по вертикальной, то пиксели по оси x остаются нетронутыми, а по оси y пиксели меняются местами относительно середины изображения. Если не существует введенной

пользователем оси, то печатается соответствующее сообщение, а изображение остается нетронутым.

2.5 Вторая функция

Вторая функция *copyImage(BMP* image, int x_src_left, int y_src_top, int x_src_right, int y_src_bottom, int x_dest_left, int y_dest_top)*. Эта функция копирую заданной область в некоторое место фотографии, которое тоже задано по координатам пользователем. Сначала идет проверка введенных координат. Если есть ошибки, то пользователь получает соответствующее сообщение. Затем, как в предыдущей функции, для удобства редактируются введенные координаты. После создается переменная типа BMP, которая хранит в себе скопированную область. Затем скопированная область вставляется на место пикселей, которые находятся по координатам назначения. В конце идет очистка памяти в двумерном массиве, который был создан для хранения скопированной области, с помощью ранее созданной функции freeMem().

2.6 Третья функция

Третья функция *changeColor(BMP* image, int r1, int r2, int g1, int g2, int b1, int b2).* Сначала идет проверка введенных компонент. Если они введены неправильно, функция прерывается, выводится соответствующая ошибка, изображение не изменяется. Затем программа проходится по всем пикселям. Если компоненты какого-то пикселя совпадают с компонентами цвета, который нужно поменять, то компоненты соответствующего пикселя меняются на компоненты цвета, на который нужно поменять. Если цвета, который нужно поменять, в изображении нет, то печатается соответствующее сообщение.

2.7 Четвертая функция

Четвертая функция *rgbFilter(BMP* image, int value, char* component)*. Функция меняет одну из компонент во всем изображении на определенное значение. Сначала идет проверка введенного значения компоненты и имени

компоненты. Если есть несоответствие, то печатается соответствующее сообщение пользователю, функция заканчивает свою работу и изображение остается без изменений. Потом программа проходится по всем пикселям и меняет заданную компоненту на заданное значение.

2.8 Функция main

Функция main содержит два аргумента: char* argv[] - массив параметров, передаваемых функции, int argc — количество параметров.

Для того, чтобы реализовать CLI, необходимо подключить библиотеку getopt.h. В самом начале записываются имя входного и выходного файла, которые являются вторым и последним параметром соответственно. После входное изображение программа пытается записать. Если это не удалось сделать, то пользователь получает сообщение об ошибке и программа завершается. Затем идет проверка на то, что второй параметр это «-h» или «--help». Если это действительно так, то печатается инструкция к программе. Затем создается массив char* opts с короткими ключами и структура struct option longOpts[] с длинными ключами. Далее для обработки и длинных, и коротких ключей используется встроенная функция getopt_long. Возвращаемое значение записывается в отдельную переменную opt. Если значение равно -1, значит нет введенных параметров и выводится инструкция пользователю.

Затем, пока возвращаемое значение переменной opt не станет -1, выполняются определенные действия, исходя из того, какой ключ ввел пользователь. Если пользователь ввел несуществующий ключ, то печатается соответствующее сообщение.

Считывание входных данных совершается с помощью функции sscanf, которая также возвращает количество удачно считанных аргументов. Если оно не соответствует количеству аргументов функции, то печатается соответствующее сообщение пользователю, а само изображение никак не меняется.

После всех преобразований BMP-файла программа пытается записать изображение. Если у нее это не получается, то печатается сообщение пользователю и происходит просто очистка выделенной памяти.

2.9 Функции вывода инструкции и информации о файле

Функции *printImageInfo(BMP image) и printHelp()*. Эти две функции функции выводят всю информацию находящуюся в полях структур, из которых состоит изображение и подробную инструкцию с примерами по использованию программы.

3. ТЕСТИРОВАНИЕ

3.1. Исходное изображение

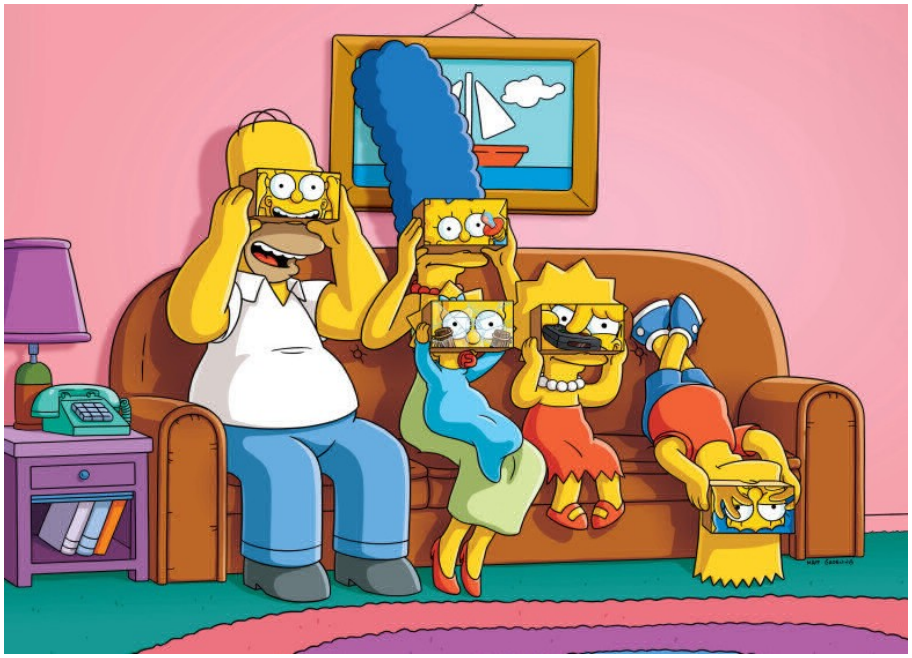

Рисунок 1: Исходное изображение

3.2. Отражение заданной области

Входные данные: ./a.out simpsonsvr.bmp -r 100,100,400,400,vertical out.bmp

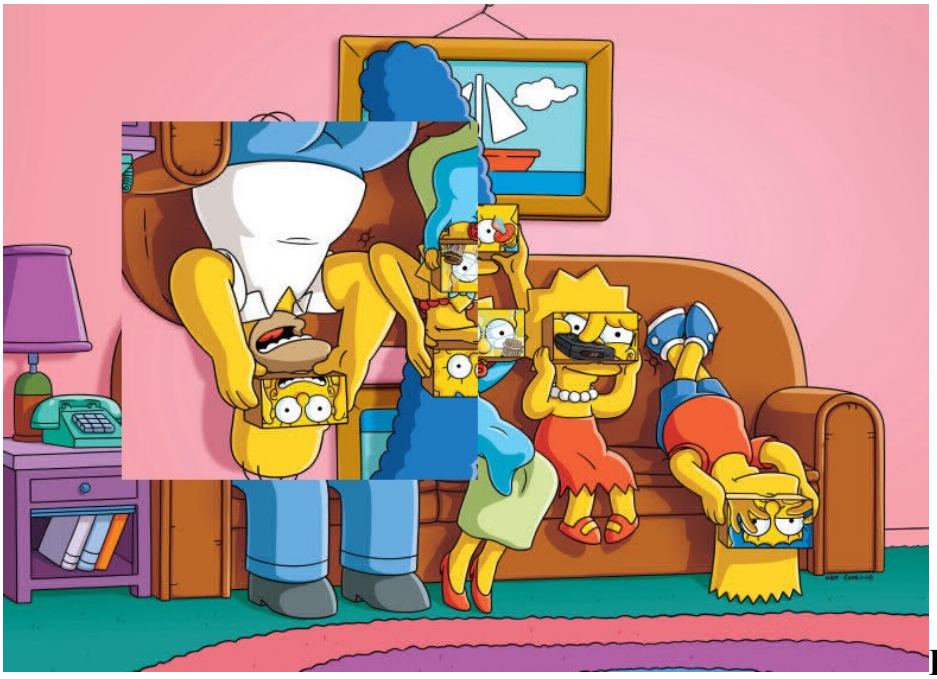

Рисунок 2: Отражение заданной области

3.3 Копирование заданной области

Входные данные: ./a.out simpsonsvr.bmp -c 100,100,200,200,200,200 out.bmp

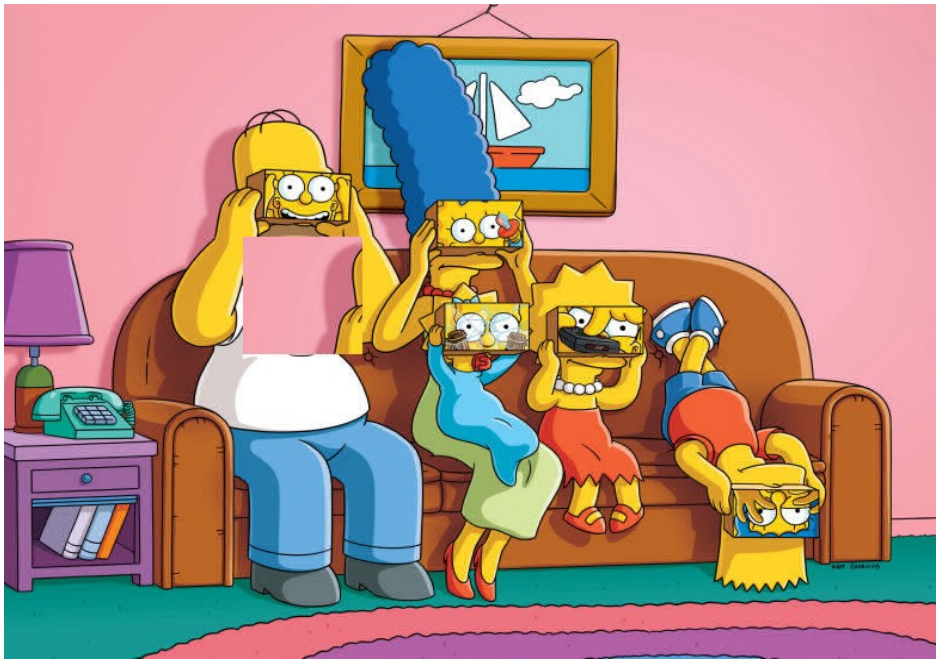

Рисунок 3: Копированиие заданной области

3.4 Замена всех пикселей одного заданного цвета на другой цвет.

Входные данные: ./a.out simpsonsvr.bmp -C 255,255,255,0,0,0 out.bmp

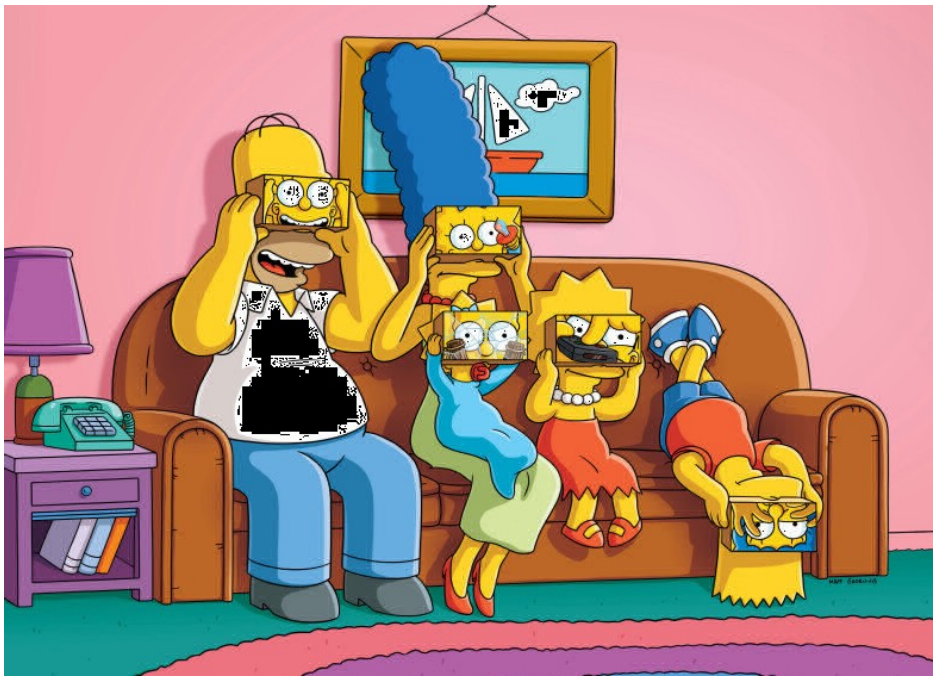

Рисунок 4: Замена всех пикселей одного цвета на другой

3.5 Фильтр rgb-компонент.

Входные данные: ./a.out simpsonsvr.bmp -f 255,red out.bmp

Рисунок 5: Фильтр rgb-компонент

3.6 Вывод информации об изображении.

Входные данные: ./a.out simpsonsvr.bmp -i

```
vladimir@vladimir-BOHK-WAX9X:~/course work$ ./a.out simpsonsvr.bmp -i
Signature:      4d42 (19778)
filesize:       141a62 (1317474)
reserved1:      0 (0)
reserved2:      0 (0)
pixelArrOffset: 36 (54)
headerSize:     28 (40)
width:          30c (780)
height:         233 (563)
planes:         1 (1)
bitsPerPixel:   18 (24)
compression:    0 (0)
imageSize:      0 (0)
xPixelsPerMeter:        2e23 (11811)
yPixelsPerMeter:        2e23 (11811)
colorsInColorTable:     0 (0)
importantColorCount:    0 (0)
```

Рисунок 6: Вывод информации об изображении

3.7 Вывод инструкции.

Входные данные: ./a.out -h

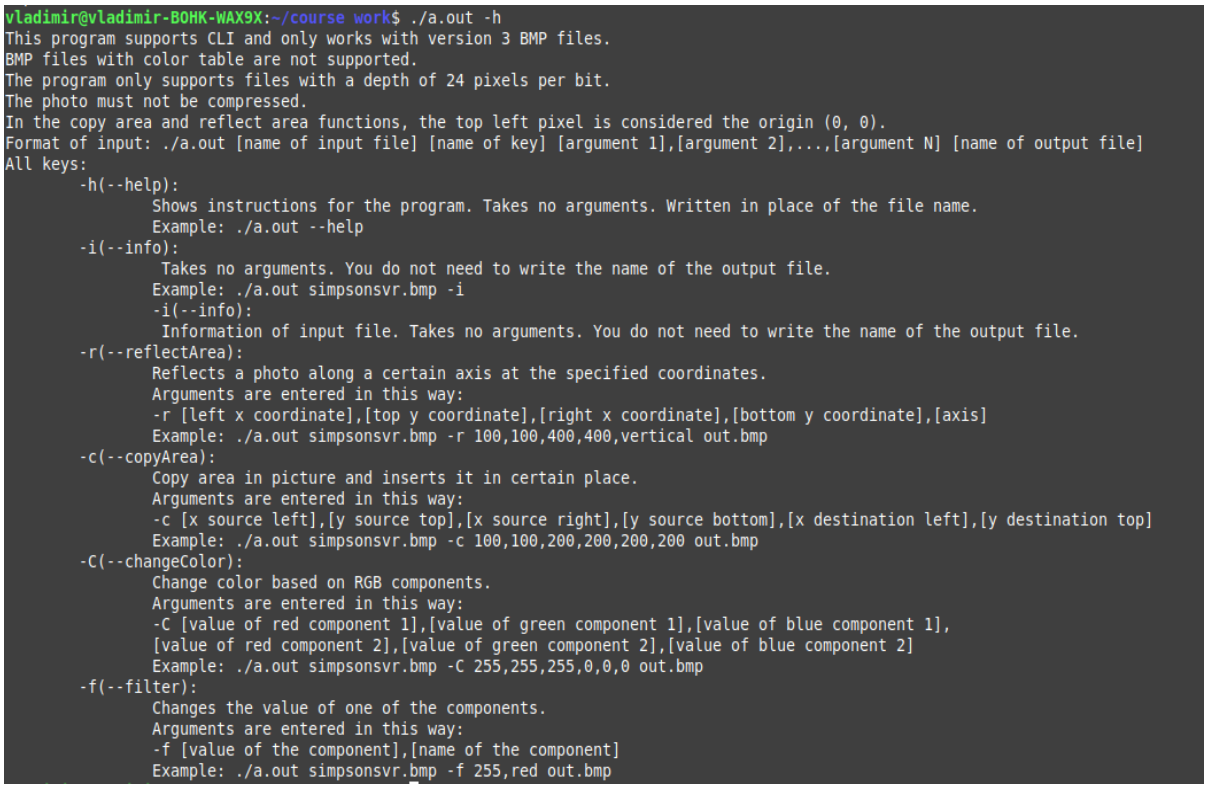

Рисунок 7: Вывод инструкции

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была создана программа, поддерживающая CLI, обрабатывающая изображение в формате BMP version 3 без сжатия, без таблицы цветов и имеющее глубину цвета 24 бита на цвет. Она способна считать изображение, записать его в другой файл, скопировать заданную область, отразить заданную область, применить rgb-фильтр компонент и заменить один цвет на другой.

Программа также обрабатывает все возможные случаи ошибок, которые могли возникнуть у пользователя из-за неправильного взаимодействия с ней.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. https://en.cppreference.com
2. https://www.cplusplus.com

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название программы: cw.c

```
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <string.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;

typedef struct
{
    BitmapFileHeader header;
    BitmapInfoHeader info;
    RGB** pixels;
}BMP;

#pragma pack(pop)


void printHelp(){
    puts("This program supports CLI and only works with version 3
BMP files.");
    puts("BMP files with color table are not supported.");
    puts("The program only supports files with a depth of 24
pixels per bit.");
    puts("The photo must not be compressed.");
    puts("In the copy area and reflect area functions, the top
left pixel is considered the origin (0, 0).");
    puts("Format of input: ./a.out [name of input file] [name of
key] [argument 1],[argument 2],...,[argument N]"
         " [name of output file]");
    puts("All keys:");
    puts("\t-h(--help):\n\t\tShows instructions for the program. "
         "Takes no arguments. Written in place of the file
name.");
    puts("\t\tExample: ./a.out --help");
    puts("\t-i(--info):\n\t\t Takes no arguments. You do not need
to write the name of the output file.");
    puts("\t\tExample: ./a.out simpsonsvr.bmp -i");
```

```
    puts("\t\t-i(--info):\n\t\t Information of input file. Takes
no arguments. "
         "You do not need to write the name of the output file.");
    puts("\t-r(--reflectArea):");
    puts("\t\tReflects a photo along a certain axis at the
specified coordinates.");
    puts("\t\tArguments are entered in this way:");
    puts("\t\t-r [left x coordinate],[top y coordinate],[right x
coordinate],[bottom y coordinate],[axis]");
    puts("\t\tExample: ./a.out simpsonsvr.bmp -r
100,100,400,400,vertical out.bmp");
    puts("\t-c(--copyArea):");
    puts("\t\tCopy area in picture and inserts it in certain
place.");
    puts("\t\tArguments are entered in this way:");
    puts("\t\t-c [x source left],[y source top],[x source right],
[y source bottom],"
         "[x destination left],[y destination top]");
    puts("\t\tExample: ./a.out simpsonsvr.bmp -c
100,100,200,200,200,200 out.bmp");
    puts("\t-C(--changeColor):");
    puts("\t\tChange color based on RGB components.");
    puts("\t\tArguments are entered in this way:");
    puts("\t\t-C [value of red component 1],[value of green
component 1],[value of blue component 1],\n"
         "\t\t[value of red component 2],[value of green component
2],[value of blue component 2]");
    puts("\t\tExample: ./a.out simpsonsvr.bmp -C 255,255,255,0,0,0
out.bmp");
    puts("\t-f(--filter):");
    puts("\t\tChanges the value of one of the components.");
    puts("\t\tArguments are entered in this way:");
    puts("\t\t-f [value of the component],[name of the
component]");
    puts("\t\tExample: ./a.out simpsonsvr.bmp -f 255,red
```

```
out.bmp");
}

void printImageInfo(BMP image){
    printf("Signature:\t%x (%hu)\n", image.header.signature,
image.header.signature);
    printf("filesize:\t%x (%u)\n", image.header.filesize,
image.header.filesize);
    printf("reserved1:\t%x (%hu)\n", image.header.reserved1,
image.header.reserved1);
    printf("reserved2:\t%x (%hu)\n", image.header.reserved2,
image.header.reserved2);
    printf("pixelArrOffset:\t%x (%u)\n",
image.header.pixelArrOffset, image.header.pixelArrOffset);
    printf("headerSize:\t%x (%u)\n", image.info.headerSize,
image.info.headerSize);
    printf("width:     \t%x (%u)\n", image.info.width,
image.info.width);
    printf("height:    \t%x (%u)\n", image.info.height,
image.info.height);
    printf("planes:    \t%x (%hu)\n", image.info.planes,
image.info.planes);
    printf("bitsPerPixel:\t%x (%hu)\n", image.info.bitsPerPixel,
image.info.bitsPerPixel);
    printf("compression:\t%x (%u)\n", image.info.compression,
image.info.compression);
    printf("imageSize:\t%x (%u)\n", image.info.imageSize,
image.info.imageSize);
    printf("xPixelsPerMeter:\t%x (%u)\n",
image.info.xPixelsPerMeter, image.info.xPixelsPerMeter);
    printf("yPixelsPerMeter:\t%x (%u)\n",
image.info.yPixelsPerMeter, image.info.yPixelsPerMeter);
    printf("colorsInColorTable:\t%x (%u)\n",
image.info.colorsInColorTable, image.info.colorsInColorTable);
    printf("importantColorCount:\t%x (%u)\
```

```
n",image.info.importantColorCount,
image.info.importantColorCount);
}

int IsImageOk(BMP image){
    if(image.info.headerSize != 40){
        puts("This version of the BMP file is not supported.
Please, change your file.");
        return 0;
    }

    if(image.info.planes != 1){
        puts("Unsupported quantity of planes. Please, change your
file.");
        return 0;
    }

    if(image.info.compression != 0){
        puts("The file is compressed. Please, change your file.");
        return 0;
    }

    if(image.info.bitsPerPixel != 24){
        puts("The color depth of the file is not 24 pixels per
bit. Please, change your file.");
        return 0;
    }

    if(image.info.colorsInColorTable != 0 ||
image.info.importantColorCount != 0){
        puts("This file uses a color table. Please, change your
file.");
        return 0;
    }
```

```
    return 1;
}

int readImage(BMP *image, char* path){
    if(path[strlen(path) - 1] != 'p' || path[strlen(path) - 2] !=
'm'
       || path[strlen(path) - 3] != 'b' || path[strlen(path) -
4] != '.'){
        puts("Wrong format of input file.");
        return 0;
    }

    FILE *f = fopen(path, "rb");

    fread(&image->header,1,sizeof(BitmapFileHeader),f);

    if(image->header.signature != 0x4d42){
        puts("The file does not conform to the BMP format. Please,
change your file.");
        return 0;
    }

    fread(&image->info,1,sizeof(BitmapInfoHeader),f);

    if(!IsImageOk(*image)){
        return 0;
    }

    unsigned int H = image->info.height;
    unsigned int W = image->info.width;
    unsigned int offset = (W*3)%4;


    image->pixels = malloc(H * sizeof(RGB*));
    if(!image->pixels){
```

```
        puts("Memory is not available.");
        return 0;
    }

    for(int i=0; i<H; i++){
        image->pixels[i] = malloc(W * sizeof(RGB) + offset);
        if(!image->pixels[i]){
            puts("Memory is not available.");
            return 0;
        }

        fread(image->pixels[i], 1,W * sizeof(RGB) + offset,f);
    }

    fclose(f);

    return 1;
}

void freeMem(BMP* image){
    for(unsigned int i = 0; i < image->info.height; ++i){
        free(image->pixels[i]);
    }
    free(image->pixels);
}

int writeImage(BMP *image, char* path){
    unsigned int H = image->info.height;
    unsigned int W = image->info.width;
    unsigned int offset = (W*3)%4;

    if(path[strlen(path) - 1] != 'p' || path[strlen(path) - 2] !=
'm'
    || path[strlen(path) - 3] != 'b' || path[strlen(path) - 4] !=
'.'){
```

```
        puts("Wrong format of output file.");
        return 0;
    }
    FILE *fout = fopen(path, "wb");


    fwrite(&image->header, 1, sizeof(BitmapFileHeader),fout);
    fwrite(&image->info, 1, sizeof(BitmapInfoHeader),fout);

    unsigned int w = W * sizeof(RGB) + offset;
    for(int i=0; i<H; i++){
        fwrite(image->pixels[i], 1, w, fout);
    }
    fclose(fout);

    freeMem(image);

    return 1;
}

void changeColor(BMP* image, int r1, int r2, int g1, int g2, int
b1, int b2){
    int checkPresenceColor = 0;
    if(r1 < 0 || r2 < 0 || g1 < 0 || g2 < 0 || b1 < 0 || b2 < 0
            || r1 > 255 || r2 > 255 || g1 > 255
            || g2 > 255 || b1 > 255 || b2 > 255){
        puts("One of the component is incorrectly set.");
        return;
    }

    for(int i = 0; i < image->info.height; ++i){
        for(int j = 0; j < image->info.width; ++j){
            if(image->pixels[i][j].r == r1 && image->pixels[i]
[j].g == g1
            && image->pixels[i][j].b == b1){
```

```
                checkPresenceColor = 1;
                image->pixels[i][j].r = r2;
                image->pixels[i][j].g = g2;
                image->pixels[i][j].b = b2;
            }
        }
    }


    if(checkPresenceColor == 0) puts("There is no this color.");

}

void swapPixels(RGB *pix1, RGB *pix2){
    RGB temp = *pix1;
    *pix1 = *pix2;
    *pix2 = temp;
}


void reflectArea(BMP* image, char* axis, int x_left, int y_top,
                 int x_right, int y_bottom){

    if(x_left < 0 || x_left > x_right || x_right > image-
>info.width
    || y_bottom > image->info.height || y_top < 0 || y_top >
y_bottom){
        puts("Wrong coordinates.");
        return;
    }

    unsigned int width_of_area = x_right - x_left;
    unsigned int height_of_area = y_bottom - y_top;

    y_bottom = (int)image->info.height - y_bottom - 1;
```

```
    y_top = (int)image->info.height - y_top;

    if(!strcmp(axis, "horizontal")){
        for (int y = 0; y < height_of_area; ++y) {
            for (int x = 0; x < width_of_area / 2; ++x) {
                swapPixels(&image->pixels[y_top - y][x_left + x],
&image->pixels[y_top - y][x_right - x]);
            }
        }
    }

    if(!strcmp(axis, "vertical")) {
        for (int y = 0; y < height_of_area / 2; ++y) {
            for (int x = 0; x < width_of_area; ++x) {
                swapPixels(&image->pixels[y_bottom + y][x_left +
x], &image->pixels[y_top - y][x_left + x]);
            }
        }
    }

    if(strcmp(axis, "vertical") != 0 && strcmp(axis, "horizontal")
!= 0){
        puts("Wrong axis selected.");
    }
}

void copyImage(BMP* image, int x_src_left, int y_src_top,
               int x_src_right, int y_src_bottom,
               int x_dest_left, int y_dest_top){

    int height = (int)image->info.height;
    int width = (int)image->info.width;


    if(x_src_left < 0 || y_src_top < 0
```

```
            || x_src_right > width || y_src_bottom > height
            || x_src_left > x_src_right || y_src_top >
y_src_bottom){
        puts("Wrong source area coordinates.");
        return;
    }

    if(x_dest_left < 0 || y_dest_top < 0){
        puts("Wrong destination area coordinates.");
        return;
    }


    y_dest_top = height - y_dest_top;
    y_src_top = height - y_src_top;
    y_src_bottom = height - y_src_bottom - 1;


    BMP newImage;

    newImage.info.height = y_src_top - y_src_bottom;
    newImage.info.width = x_src_right - x_src_left;

    unsigned int y_dest_bottom = y_dest_top -
newImage.info.height;
    unsigned int x_dest_right = x_dest_left + newImage.info.width;

    if(y_dest_bottom > height || x_dest_right > width){
        puts("Wrong destination area coordinates.");
        return;
    }


    newImage.pixels = malloc(newImage.info.height * sizeof(RGB*));
```

```
    for(int i = 0; i < newImage.info.height; ++i){
        newImage.pixels[i] = malloc(newImage.info.width *
sizeof(RGB));
    }

    unsigned int x;
    unsigned int y = 0;

    for(unsigned int i = y_src_bottom; i < y_src_top; ++i){
        x = 0;
        for(unsigned int j = x_src_left; j < x_src_right; ++j){
            newImage.pixels[y][x] = image->pixels[i][j];
            ++x;
        }
        ++y;
    }

    y = 0;
    x = 0;


    for(unsigned int i = y_dest_bottom; i < y_dest_top; ++i){
        for(unsigned int j = x_dest_left; j < x_dest_right; ++j){
            image->pixels[i][j] = newImage.pixels[y][x];
            ++x;
        }
        ++y;
        x = 0;
    }

    freeMem(&newImage);
}

void rgbFilter(BMP* image, int value, char* component){
    if(value < 0 || value > 255){
```

```
        puts("Invalid component value.");
        return;
    }

    if(strcmp(component, "red") != 0 && strcmp(component, "green")
!= 0
    && strcmp(component, "blue") != 0){
        printf("Wrong component name.");
        return;
    }

    for(int i = 0; i < image->info.height; ++i){
        for(int j = 0; j < image->info.width; ++j){
            if(!strcmp(component, "red"))
                image->pixels[i][j].r = value;

            if(!strcmp(component, "green"))
                image->pixels[i][j].g = value;

            if(!strcmp(component, "blue"))
                image->pixels[i][j].b = value;
        }
    }
}

int main(int argc, char* argv[]){
    BMP image;

    char filename[50];
    char out_file[50];
    strcpy(filename, argv[1]);
    strcpy(out_file, argv[argc - 1]);

    if(!strcmp(filename, "-h") || !strcmp(filename, "--help")){
        printHelp();
```

```
    return 0;
}

char *opts = "r:c:f:C:ih";

struct option longOpts[]={
        {"reflect", required_argument, NULL, 'r'},
        {"copy", required_argument, NULL, 'c'},
        {"filter", required_argument, NULL, 'f'},
        { "changeColor", required_argument, NULL, 'C'},
        { "info", no_argument, NULL, 'i'},
        { "help", no_argument, NULL, 'h'}
};
int opt;
int longIndex;
opt = getopt_long(argc, argv, opts, longOpts, &longIndex);

if(!readImage(&image, filename)) return 0;


if(opt == -1){
    puts("Invalid input format.");
    return 0;
}

while (opt != -1) {

    switch(opt){
        case 'h': {
            printHelp();
            freeMem(&image);

            return 0;
        }
```

```
            case 'i': {
                printImageInfo(image);
                freeMem(&image);

                return 0;
            }

            case 'r': {
                char string[50];
                int x_left, x_right, y_top, y_bottom;

                int count = sscanf(optarg, "%d,%d,%d,%d,%s",
&x_left, &y_top, &x_right, &y_bottom, string);

                if(count < 5){
                    puts("Too few arguments to do this function (-
r/--reflect).");
                    break;
                }

                reflectArea(&image, string, x_left, y_top,
x_right, y_bottom);
                break;
            }

            case 'c': {
                int x_src_left, x_src_right, y_src_top,
y_src_bottom, x_dest_left, y_dest_top;

                int count = sscanf(optarg, "%d,%d,%d,%d,%d,%d",
&x_src_left, &y_src_top, &x_src_right,
                                    &y_src_bottom, &x_dest_left,
&y_dest_top);

                if(count < 6){
```

```
                puts("Too few arguments to do this function (-
c/--copy).");
                break;
            }

            copyImage(&image, x_src_left, y_src_top,
x_src_right, y_src_bottom, x_dest_left, y_dest_top);
            break;
        }

        case 'C':{
            int r1, g1, b1, r2, g2, b2;

            int count = sscanf(optarg, "%d,%d,%d,%d,%d,%d",
&r1, &g1, &b1,
                                    &r2, &g2, &b2);

            if(count < 6){
                puts("Too few arguments to do this function (-
C/--changeColor).");
                break;
            }

            changeColor(&image, r1, r2, g1, g2, b1, b2);
            break;
        }

        case 'f': {
            int value;
            char str[50];

            int count = sscanf(optarg, "%d,%s", &value, str);

            if(count < 2){
                puts("Too few arguments to do this function (-
```

```
f/--filter).");
                    break;
                }

                rgbFilter(&image, value, str);
                break;
            }

            default: {
                puts("No such key.");
                break;
            }

        }
        opt = getopt_long(argc, argv, opts, longOpts, &longIndex);
    }

    if(!writeImage(&image, out_file)){
        freeMem(&image);
    }


    return 0;
}
```