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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка изображений

Студент гр. 3342 _____ Белаид Фарук

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2024

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Белаид Фарук

Группа 3342

Тема работы Обработка изображений

Исходные данные :

Вариант 3.2

Содержание пояснительной записки:

Программа **обязательно должна иметь CLI** (опционально дополнительное использование GUI).

Программа должна реализовывать весь следующий функционал по обработке bmp-файла

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 иметь следующую функции по обработке изображений:

- Инверсия цвета на всём изображении. Флаг для выполнения данной операции: `--inverse`
- Установить компоненту цвета, как сумму двух других. Флаг для

выполнения данной операции: `--component_sum`. Функционал определяется

- Какую компоненту требуется изменить. Флаг `--component_name`. Возможные значения `red`, `green` и `blue`.

Разделы пояснительной записки : «Содержание», «Введение», «Ввод и вывод изображения», «Обработка изображения», «Заключение», «Список использованных источников».

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 22.05.2024

Дата защиты реферата: 29.05.2024

Студент _____ Белаид Фарук

Преподаватель _____ Глазунов С.А.

АННОТАЦИЯ

Курсовая работа представляет собой программу на языке Си, обрабатывающую BMP изображение по тексту задания.

Для работы с изображением использовались функции стандартных библиотек, динамическая память, структуры.

В рамках данной курсовой работы была реализована программа, имеющая следующий функционал:

1. Инверсия цвета на всём изображении.
2. Установить компоненту цвета, как сумму двух других.

SUMMARY

The course work is a C program that processes a BMP image based on the text of the assignment.

The functions of standard libraries, dynamic memory, and structures were used to work with the image.

As part of this course work, a program was implemented that has the following functionality:

1. Inversion of the color in the entire image.
2. Set the color component as the sum of the other two.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной работы является разработка приложения с CLI по обработке изображения формата BMP.

На основе вышеизложенной цели, были сформулированы следующие задачи :

1. Изучение особенностей работы с *getopt_long*

2. Изучение строения и особенностей хранения BMP файлов в памяти компьютера

3. Создание CLI

1. ВВОД И ВЫВОД ИЗОБРАЖЕНИЯ

1.1 Ввод изображения

Функция loadBMPImg загружает BMP изображение из файла и возвращает структуру BMPFile, содержащую данные изображения. Она открывает файл для чтения, считывает заголовок и информацию о изображении, затем выделяет память для пиксельных данных. Функция также учитывает смещение в строках пикселей, необходимое для выравнивания данных. Если файл не найден или не удается выделить память, функция выводит сообщение об ошибке и завершает программу.

1.2 Вывод изображения

Функция saveBMP сохраняет изображение BMP в указанный файл. Она открывает файл в режиме записи бинарных данных и проверяет успешность открытия, выводя сообщение об ошибке и завершая программу в случае неудачи. Затем вычисляет смещение для выравнивания строк изображения по границе в 4 байта, записывает заголовок файла (BitmapFileHeader) и информационный заголовок (BitmapInfoHeader) в файл. После этого записывает пиксельные данные построчно, включая смещение для каждой строки. В конце функция закрывает файл. Эта функция позволяет сохранить измененное или новое изображение BMP на диск.

2. ОБРАБОТКА ИЗОБРАЖЕНИЯ

2.1 Инверсии цветов на всем изображении

Запуск программы с помощью клавиши **–i –inverse** вызывает функцию **invertColors(),** Эта функция принимает на вход указатель на изображение в формате BMP и инвертирует цвета всех пикселей этого изображения. Она использует функцию **invertPixels** для инвертирования каждого пикселя. Два вложенных цикла проходят по всем пикселям изображения (высота и ширина), инвертируя цвет каждого пикселя.

2.2 Установка цветового компонента как суммы двух других

Выполнение программы с флагом **–s –component_sum** запускает функцию **changeComponentColor(),** Эта функция изменяет цветовой компонент всех пикселей изображения на сумму двух других компонентов. В зависимости от значения pixelComponent («red», «green», «blue»), она определяет, какой компонент цвета изменять:

- Если pixelComponent равен «red», красный компонент каждого пикселя устанавливается как сумма зеленого и синего компонентов.
- Если pixelComponent равен «green», зеленый компонент каждого пикселя устанавливается как сумма синего и красного компонентов.
- Если pixelComponent равен «blue», синий компонент каждого пикселя устанавливается как сумма красного и зеленого компонентов.

Если значение pixelComponent не соответствует ни одному из указанных значений, выводится ошибка и программа завершается.

2.3 Измените старый цвет компонента на новый, используя определенную формулу

Выполнение программы с флагом **-g --gamma** запускает функцию **ChangeGamma()** которая применяет коррекцию гаммы ко всему изображению BMP, проходя по каждому пикселю и изменяя значения его цветовых

компонентов (красного, зеленого и синего) в соответствии с новой гаммой, используя функцию **CalculateNewValue**.

3. ОСТАЛЬНЫЕ ФУНКЦИЯ

3.1. Функция main

Для корректной работы программы подключены стандартные библиотеки языка Си: stdlib.h, stdio.h, math.h.

Также подключена библиотека getopt.h для анализа аргументов командной строки.

Функция main обрабатывает аргументы командной строки для выполнения различных операций с BMP-изображением. Она начинается с проверки достаточного количества аргументов и предоставляет инструкции по использованию, если это необходимо. Функция поддерживает опции для отображения информации о BMP-файле, инвертирования цветов, изменения определенных цветовых компонентов и применения гамма-коррекции. Она анализирует параметры командной строки с помощью getopt_long, проверяет корректность входных и выходных имен файлов, гарантирует, что запрашивается только одна операция за раз, и затем выполняет запрошенную операцию с BMP-файлом. После обработки она сохраняет измененное BMP-изображение в указанный выходной файл и освобождает выделенную память перед завершением работы.

3.2 Функции print

Функция printHelp выводит имя студент, вариант курсовой работы и справку о возможных аргументах.

Функции printFileHeader и printInfoHeader выводят информацию о файле: ширину изображения, высоту изображения, тип цвета и глубину цвета.

3.3 Дополнительные функции

Функция **invertPixels** принимает структуру RGB в качестве аргумента и инвертирует её цветовые компоненты. Для каждого из каналов (красный, зеленый и синий) она вычитает текущее значение из максимального возможного значения RGBMAX, тем самым изменяя цвет на противоположный. После инвертирования всех компонентов функция возвращает обновлённую структуру RGB.

Функция **sumOfTwoComponents** принимает два значения цветовых компонентов в виде unsigned char и вычисляет их сумму. Если результат сложения не превышает 255, то он возвращается как есть; в противном случае возвращается 255. Это гарантирует, что результат всегда остаётся в пределах допустимого диапазона для цветовых компонентов.

Функция **CalculateNewValue** вычисляет новое значение цвета, применяя гамма-коррекцию к исходному значению цвета. Она принимает исходное значение цвета old Color Value и значение гаммы value, нормализует исходное значение к диапазону [0, 1], применяет функцию возведения в степень, а затем масштабирует результат обратно в диапазон [0, 255]. Окончательное значение округляется вниз до ближайшего целого числа.

Функция void **freeBMPFile** освобождает память, выделенную для хранения данных изображения BMP. Для каждой строки изображения, находящейся в массиве img.data, выполняется освобождение памяти, после чего освобождается память, выделенная под сам массив img.data.

Функция int **calculateOffset** вычисляет и возвращает величину смещения в байтах, необходимого для выравнивания строки изображения BMP по 4 байта.

Функция int **checkBmpFile** проверяет корректность формата BMP файла. Если файл не соответствует ожидаемым критериям, функция выводит сообщение об ошибке и завершает выполнение программы с соответствующим кодом ошибки.

ЗАКЛЮЧЕНИЕ

В ходе курсовой работы были изучены особенности строения и хранения BMP файлов в памяти компьютера. Освоена работа с BMP файлами на языке программирования C и работа с CLI.

Была написана программа с CLI для обработки изображений формата BMP.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Правила оформления пояснительной записки к курсовой работе:
https://se.moevm.info/doku.php/courses:programming:report
2. Теоретические сведения о bmp:
https://en.wikipedia.org/wiki/BMP_file_format
3. Getopt Manual: [Электронный ресурс].
URL: https://man7.org/linux/man-pages/man3/getopt.3.html

ПРИЛОЖЕНИЕ А
ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

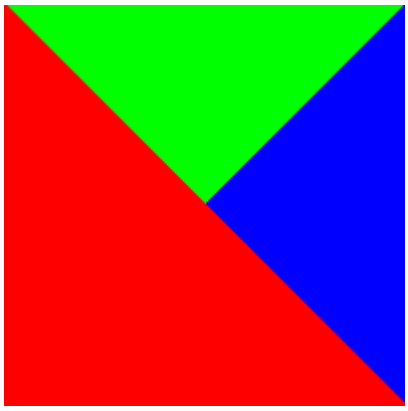

Рисунок 1 исходное изображение

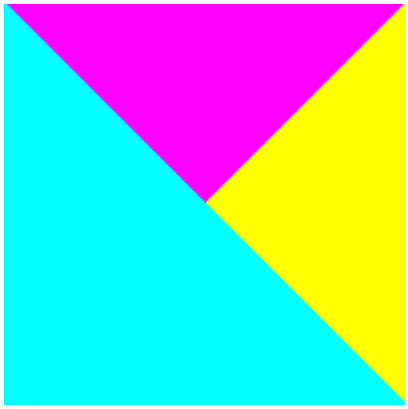

Рисунок 2 результат работы функции invertColors

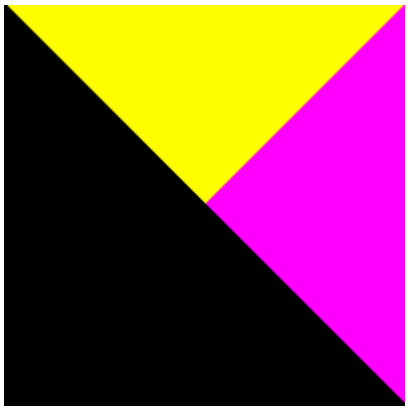

Рисунок 3 результат работы функции changeComponentColor

Рисунок 4 исходное изображение

Рисунок 5 результат работы функции ChangeGamma

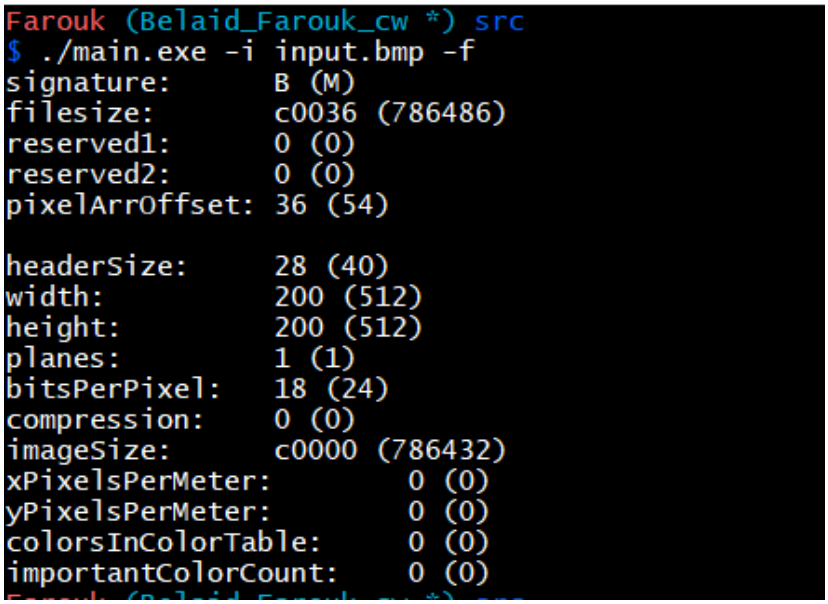

Рисунок 6 Тестирование вывода информации о файле.

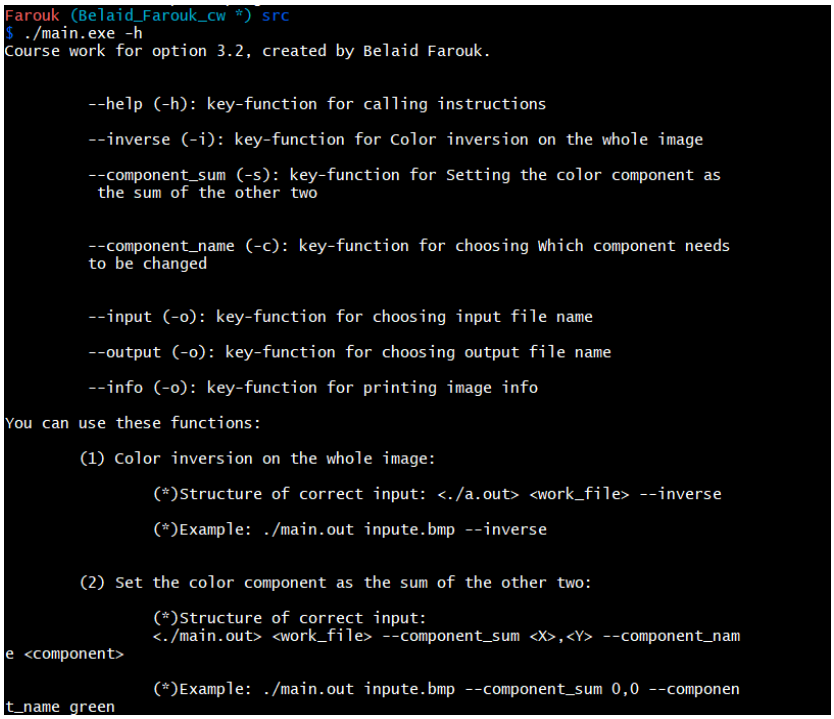

Рисунок 7 Тестирование вывода справки.

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

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

#define FILE_NOT_FOUND_ERROR 40

#define WRITE_FILE_ERROR 42

#define INCORRECT_BMP_FORMAT_ERROR 43

#define INSUFFICIENT_ARGUMENTS_ERROR 44

#define MEMORY_ALLOCATION_ERROR 45

#define UNKNOWN_OPTION_ERROR 46

#define INVALID_OPTION_ARG_ERROR 47

#define RGBMAX 255

#pragma pack(push, 1)

typedef struct {
    unsigned char ID[2] ;
    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)

typedef struct BMPFile {
    BitmapFileHeader header;
    BitmapInfoHeader info;
    RGB** data;
} BMPFile;

void printFileHeader(BitmapFileHeader header);
void printInfoHeader(BitmapInfoHeader header);
int calculateOffset(unsigned int width);

BMPFile loadBMPImg(char* fileName);
int checkBmpFile(BMPFile* img);
void saveBMP(char* fileName, BMPFile img);

RGB invertPixels(RGB pixel);
void invertColors(BMPFile* img);

int sumOfTwoComponents(unsigned char firstPixel,unsigned char
secondPixel);
void changeComponentColor(BMPFile* img, const char* pixelComponent);

// void printDetails(BMPFile* img);
int CalculateNewValue(int oldColorValue,float value);
void ChangeGamma(BMPFile* img, float value);

void freeBMPFile(BMPFile img);
void printHelp();

int main(int argc, char* argv[]) {
    if (argc < 3){
        if (argc < 2){
            puts("Not enough arguments to run the program. Use -h (--
help) key to read the instructions.");
            exit(INSUFFICIENT_ARGUMENTS_ERROR);
        }

        else if (strcmp(argv[1], "--help") == 0 || strcmp(argv[1], "-h")
== 0) {
            printHelp();
            exit(EXIT_SUCCESS);
        }

        puts("Incorrect key. Use -h (--help) key to read the
instructions.");
        exit(UNKNOWN_OPTION_ERROR);
    }

    const char* short_options = "c:o:i:v:hfIsg";

    const struct option long_options[] = {
           { "help", no_argument, NULL, 'h' },
        { "info", no_argument, NULL, 'f' },
           { "inverse", no_argument, NULL, 'I' },
           { "component_sum", no_argument, NULL, 's' },
           { "component_name", required_argument, NULL, 'c' },
```

```
            { "gamma", no_argument, NULL, 'g' },
            { "value", required_argument, NULL, 'v' },

            { "output", required_argument, NULL, 'o' },
            { "input", required_argument, NULL, 'i' },
            { NULL, 0, NULL, 0 }
      };

    int opt;
     int optIndex;
    char* inputFileName = "";
    char* outputFileName = "";
    int invertColorFlag = 0;
    int componentSumFlag = 0;
    int infoFlag = 0;
    char* componentToChange = "";

    int gammaFlag = 0;
    float gammaValue = -1.0;

    opterr = 0;

     opt = getopt_long(argc, argv,short_options, long_options,
&optIndex);

    while(opt != -1){
        switch (opt){
            case 'h':
                puts("Error: Too many arguments for 'help' function");
                exit(UNKNOWN_OPTION_ERROR);
            case 'f':
                infoFlag = 1;
                break;
            case 'I':
                invertColorFlag = 1;
                break;
            case 's':
                componentSumFlag = 1;
                break;
            case 'c':
                componentToChange = optarg;
                break;
            case 'g':
                gammaFlag = 1;
                break;
            case 'v':
                sscanf(optarg,"%f",&gammaValue);
                break;
            case 'o':
                outputFileName = optarg;
                break;
            case 'i':
                inputFileName = optarg;
                break;
            case '?': default: {
                      printf("found unknown option\n");
```

```
                    exit(UNKNOWN_OPTION_ERROR);
                };
        }
        opt = getopt_long(argc, argv,short_options, long_options,
&optIndex);
    }

    if(strcmp(inputFileName, "") == 0){
        printf("no file name\n");
        exit(FILE_NOT_FOUND_ERROR);
    }

    if(strcmp(inputFileName, outputFileName) == 0){
        printf("Input/Output FileName are the same\n");
        exit(WRITE_FILE_ERROR);
    }

    // CHECK HOW MANY REQUASTED OPERATION
    if((invertColorFlag && componentSumFlag) || (invertColorFlag &&
gammaFlag) || (componentSumFlag && gammaFlag)){
       puts("Error : too many operation. please choose one operation at
time\n");
       exit(UNKNOWN_OPTION_ERROR);
    }

    // LOAD BMP FILE
    BMPFile NewBmpFile = loadBMPImg(inputFileName);

    // CHECK VALIDITY OF BMP FILE
    if (checkBmpFile(&NewBmpFile)){
        printf("BMP FILE CHECK FAILS\n");
        exit(INCORRECT_BMP_FORMAT_ERROR);
    }

    if(infoFlag){
        printFileHeader(NewBmpFile.header);
        printInfoHeader(NewBmpFile.info);
        return 0;
    }

    if(invertColorFlag){
        invertColors(&NewBmpFile);
    }

    if (componentSumFlag){
        if(strcmp(componentToChange, "") != 0){
            changeComponentColor(&NewBmpFile,componentToChange);
        }else{
            puts("Error: Missing argument for --component_name\n");
            freeBMPFile(NewBmpFile);
            exit(INVALID_OPTION_ARG_ERROR);
        }
    }

    if(gammaFlag){
        if(gammaValue > 0){
            ChangeGamma(&NewBmpFile,gammaValue);
```

```
        }else{
            puts("Error: Missing argument for --value\n");
            freeBMPFile(NewBmpFile);
            exit(INVALID_OPTION_ARG_ERROR);
        }
    }

    // SAVE BMP FILE
    if(strcmp(outputFileName, "") != 0){
        saveBMP(outputFileName, NewBmpFile);
    }else{
        saveBMP("out.bmp", NewBmpFile);
    }

    freeBMPFile(NewBmpFile);
    return 0;
}

void printFileHeader(BitmapFileHeader header) {
    printf("signature:\t%c (%c)\n", header.ID[0], header.ID[1]);
    printf("filesize:\t%x (%u)\n", header.filesize, header.filesize);
    printf("reserved1:\t%x (%hu)\n", header.reserved1, header.reserved1);
    printf("reserved2:\t%x (%hu)\n", header.reserved2, header.reserved2);
    printf("pixelArrOffset:\t%x (%u)\n", header.pixelArrOffset,
header.pixelArrOffset);
}

void printInfoHeader(BitmapInfoHeader header) {
    printf("\nheaderSize:\t%x (%u)\n", header.headerSize,
header.headerSize);
    printf("width:     \t%x (%u)\n", header.width, header.width);
    printf("height:    \t%x (%u)\n", header.height, header.height);
    printf("planes:    \t%x (%hu)\n", header.planes, header.planes);
    printf("bitsPerPixel:\t%x (%hu)\n", header.bitsPerPixel,
header.bitsPerPixel);
    printf("compression:\t%x (%u)\n", header.compression,
header.compression);
    printf("imageSize:\t%x (%u)\n", header.imageSize, header.imageSize);
    printf("xPixelsPerMeter:\t%x (%u)\n", header.xPixelsPerMeter,
header.xPixelsPerMeter);
    printf("yPixelsPerMeter:\t%x (%u)\n", header.yPixelsPerMeter,
header.yPixelsPerMeter);
    printf("colorsInColorTable:\t%x (%u)\n", header.colorsInColorTable,
header.colorsInColorTable);
    printf("importantColorCount:\t%x (%u)\n", header.importantColorCount,
header.importantColorCount);
}

int calculateOffset(unsigned int width) {
    int offset = (width*3)%4;
    offset = offset ? 4-offset : 0;
    return offset;
}

BMPFile loadBMPImg(char* fileName){
    FILE *BmpImg = fopen(fileName, "rb");
```

```
    if (BmpImg == NULL){
        puts("Error: file not found.");
        exit(FILE_NOT_FOUND_ERROR);
    }

    BMPFile NewBmpImg;

    fread(&NewBmpImg.header,1,sizeof(BitmapFileHeader),BmpImg);
    fread(&NewBmpImg.info,1,sizeof(BitmapInfoHeader),BmpImg);

    unsigned int height = NewBmpImg.info.height;
    unsigned int width = NewBmpImg.info.width;

    int offset = calculateOffset(width);

    NewBmpImg.data = malloc(sizeof(RGB*)*height);

    if (NewBmpImg.data == NULL) {
        printf("Error: cannot allocate memory!\n");
        exit(MEMORY_ALLOCATION_ERROR);
    }

    for(int i=0; i<height; i++) {
        NewBmpImg.data[i] = malloc(width * sizeof(RGB) + offset);
        if (NewBmpImg.data[i] == NULL) {
            printf("Error: cannot allocate memory!\n");
            exit(MEMORY_ALLOCATION_ERROR);
        }
        fread(NewBmpImg.data[i],1,width * sizeof(RGB) + offset,BmpImg);
    }

    fclose(BmpImg);
    return NewBmpImg;
}

int checkBmpFile(BMPFile* img){
    if(img->header.ID[0] != 'B' || img->header.ID[1] !=
'M'){puts("Incorrect signature format (not 'BM').");
exit(INCORRECT_BMP_FORMAT_ERROR);}

    if(img->info.headerSize != 40){puts("Error: Incorrect header size.");
exit(INCORRECT_BMP_FORMAT_ERROR);}

    if(img->header.pixelArrOffset != 54){puts("Error: Incorrect pixels'
offset."); exit(INCORRECT_BMP_FORMAT_ERROR);}

    if(img->info.width == 0){puts("Error: Unable to work with image (zero
width size)."); exit(INCORRECT_BMP_FORMAT_ERROR);}

    if(img->info.height == 0){puts("Error: Unable to work with image
(zero height size)."); exit(INCORRECT_BMP_FORMAT_ERROR);}

    if(img->info.planes != 1){puts("Error: Incorrect color planes (only 1
is available)."); exit(INCORRECT_BMP_FORMAT_ERROR);}
```

```
    if(img->info.bitsPerPixel != 24){puts("Error: Incorrect BMP format
(only 24-bits per pixel format is available).\nUse -h (--help) keys to
watch the instructions."); exit(INCORRECT_BMP_FORMAT_ERROR);}

    if(img->info.compression != 0){puts("Error: Function doesn't work
with compressed BMP files."); exit(INCORRECT_BMP_FORMAT_ERROR);}

    return 0;
}

void saveBMP(char* fileName, BMPFile img) {
    FILE* fp = fopen(fileName, "wb");

    if(!fp) {
        puts("Error: something went wrong while creating/opening file.");
        exit(WRITE_FILE_ERROR);
    }

    int offset = calculateOffset(img.info.width);

    fwrite(&img.header, 1, sizeof(BitmapFileHeader), fp);
    fwrite(&img.info, 1, sizeof(BitmapInfoHeader), fp);

    size_t height = img.info.height;
    size_t width = img.info.width;

    for(size_t i = 0; i < height; i++) {
        fwrite(img.data[i], 1, sizeof(RGB) * width + offset, fp);
    }

    fclose(fp);
}

RGB invertPixels(RGB pixel) {
    pixel.b = RGBMAX - pixel.b;
    pixel.g = RGBMAX - pixel.g;
    pixel.r = RGBMAX - pixel.r;

    return pixel;
}

void invertColors(BMPFile* img) {
    for(int i = 0; i < img->info.height; i++) {
        for(int j = 0; j < img->info.width; j++) {
            // Invert the RGB values of each pixel
            img->data[i][j] = invertPixels(img->data[i][j]);
        }
    }
}

int sumOfTwoComponents(unsigned char firstPixel,unsigned char
secondPixel) {
    int res = firstPixel + secondPixel;
    return res <= 255 ? res : 255;
}

void changeComponentColor(BMPFile* img, const char* pixelComponent) {
```

```
    if (strcmp(pixelComponent, "red") == 0) {
        for(int i = 0; i < img->info.height; i++) {
            for(int j = 0; j < img->info.width; j++) {
                img->data[i][j].r = sumOfTwoComponents(img-
>data[i][j].g,img->data[i][j].b);
            }
        }
    } else if (strcmp(pixelComponent, "green") == 0) {
        for(int i = 0; i < img->info.height; i++) {
            for(int j = 0; j < img->info.width; j++) {
                img->data[i][j].g = sumOfTwoComponents(img-
>data[i][j].b,img->data[i][j].r);
            }
        }

    } else if (strcmp(pixelComponent, "blue") == 0) {
        for(int i = 0; i < img->info.height; i++) {
            for(int j = 0; j < img->info.width; j++) {
                img->data[i][j].b = sumOfTwoComponents(img-
>data[i][j].r,img->data[i][j].g);
            }
        }
    } else {
        printf("ERROR: Unknown color\n");
        exit(INVALID_OPTION_ARG_ERROR);
    }
}

// void printDetails(BMPFile* img){
//     for(int i =0; i < img->info.height; i++){
//         for(int j =0; j < img->info.width;j++){
//             printf("%d %d %d\n",img->data[i][j].r,img-
>data[i][j].g,img->data[i][j].b);
//         }
//         printf("\n");
//     }
// }

int CalculateNewValue(int oldColorValue,float value){
    return (int)floor(pow((oldColorValue / 255.0),value) * 255.0);
}

void ChangeGamma(BMPFile* img, float value){
    for(int i =0; i < img->info.height; i++){
        for(int j =0; j < img->info.width;j++){
            img->data[i][j].r = CalculateNewValue(img-
>data[i][j].r,value);
            img->data[i][j].g = CalculateNewValue(img-
>data[i][j].g,value);
            img->data[i][j].b = CalculateNewValue(img-
>data[i][j].b,value);
        }
    }
}

void freeBMPFile(BMPFile img){
    for (size_t i = 0; i < img.info.height; i++){
```

```
        free(img.data[i]);
    }
    free(img.data);
}

void printHelp(){
    puts("Course work for option 3.2, created by Belaid Farouk.\n\n");
    puts("\t --help (-h): key-function for calling instructions\n");
    puts("\t --inverse (-i): key-function for Color inversion on the
whole image\n");
    puts("\t --component_sum (-s): key-function for Setting the color
component as \n \t  the sum of the other two\n\n");
    puts("\t --component_name (-c): key-function for choosing Which
component needs\n \t to be changed\n\n");
    puts("\t --input (-o): key-function for choosing input file name\n");
    puts("\t --output (-o): key-function for choosing output file
name\n");
    puts("\t --info (-o): key-function for printing image info\n");

    puts("You can use these functions:\n");

    puts("\t(1) Color inversion on the whole image:\n");
    puts("\t\t(*)Structure of correct input: <./a.out> <work_file> --
inverse\n");
    puts("\t\t(*)Example: ./main.out inpute.bmp --inverse\n\n");

    puts("\t(2) Set the color component as the sum of the other two:\n");
    puts("\t\t(*)Structure of correct input: \n\t\t<./main.out>
<work_file> --component_sum <X>,<Y> --component_name <component>\n");
    puts("\t\t(*)Example: ./main.out inpute.bmp --component_sum 0,0 --
component_name green\n\n");

}
```