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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка BMP-изображений

Вариант 6

Студентка гр. 1382 _____ Рымарь М.И.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2022

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студентка Рымарь М.И.

Группа 1382

Тема работы: Обработка BMP-файлов

Исходные данные:
Программа принимает на вход аргументы и изображение в формате BMP. Необходимо преобразовать картинку в соответствии с условиями и сохранить изменившуюся копию. Поддержка ведётся через терминальный интерфейс (CLI – Command Line Interface).

Предполагаемый объем пояснительной записки:
Не менее 15 страниц.

Дата выдачи задания: 22.03.2022

Дата сдачи реферата: 28.08.2022

Дата защиты реферата: 01.09.2022

Студентка _____ Рымарь М.И.

Преподаватель _____ Жангиров Т.Р.

АННОТАЦИЯ

В ходе выполнения курсовой работы была создана программа на языке программирования C, которая обрабатывает BMP-файл. Программа имеет CLI (Command Line Interface) с помощью библиотеки getopt.h. CLI даёт возможность вывода справки о программе, информации о файле, а также реализуемых функциях, их аргументах и примерах ввода. Программа поддерживает BMP-файлы третьей версии, глубиной кодирования 24 бита, без сжатия.

SUMMARY

During the course work, a program was developed in the C programming language that processes a BMP file. The program has a CLI (Command Line Interface) using the getopt.h library. The CLI makes it possible to output help about the program, information about the file, as well as implemented functions, their arguments and input examples. The program supports BMP files of the third version, encoding depth of 24 bits, without compression.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Кратко описать цель работы, основные задачи и методы их решения. Требуется написать программу, которая производит выбранную пользователем обработку изображения.

Программа разработана на базе ОС Windows 11 в IDE Clion. Тестирование велось в терминале Ubuntu, встроенном в IDE.

Программа предоставляет CLI – терминальный интерфейс. Все случаи некорректного ввода команд отлавливаются; выводятся соответствующие ошибки, предупреждения и рекомендации.

1.1. Задание курсовой работы

Вариант 6

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 следующий функционал по обработке bmp-файла:

(1) Рисование отрезка. Отрезок определяется:

координатами начала

координатами конца

цветом

толщиной

(2) Инвертировать цвета в заданной окружности. Окружность определяется либо координатами левого верхнего и правого нижнего угла квадрата, в который она вписана, либо координатами ее центра и радиусом.

(3) Обрезка изображения. Требуется обрезать изображение по заданной области. Область определяется:

Координатами левого верхнего угла

Координатами правого нижнего угла

1.2. Цель и задачи работы

Цель: ознакомиться с особенностями работы с BMP файлами и написать программу, которая выполняет определённую обработку изображений, заданную пользователем с помощью Command Line Interface.

Задачи:

- Реализовать корректное считывание и хранение bmp-файла;
- Осуществить правильную обработку запросов пользователя, используя CLI;
- Выполнить обработку возможных ошибок;
- Создать следующие функции: рисование отрезка, инвертирование цветов в заданной окружности и обрезка изображения.

2. ХОД РАБОТЫ

2.1. Заголовочные файлы

Для корректной работы программы подключаем следующие библиотеки:

1) <*getopt.h*> - обеспечивает command line interface, удобный для пользователя. Функции *getopt* и *getopts_long* автоматизируют часть рутинной работы, связанной с анализом типичных параметров командной строки;
2) <*stdio.h*> - заголовочный файл, который обеспечивает считывание данных, введённых пользователем, и работу с самим bmp-файлом (используемые функции*: fopen(), fclose(), fread(), fwrite(), puts(), scanf(), printf(), sscanf()*);
3) <*stdlib.h*> - заголовочный файл, который содержит в себе функции выделения памяти (используемые функции: *malloc(), free()*);
4) <*string.h*> - заголовочный файл для работы со строками (используемая функция – *strlen()*);
5) <*math.h*> - заголовочный файл, который используется для выполнения простых математических операций (используемая функция – *fabs()*).

2.2. Структуры

Все структуры оборачиваем в *#pragma pack(push, 1)* и *#pragma pack(pop),* где первый макрос пишем до структур, второй - после.

Первое устанавливает размер выравнивания в 1 байт, второе возвращает предыдущую настройку. Без этого размер структур будет меняться в зависимости от компилятора.

Далее создаём структуры *BitmapFileHeader* (хранит поля *signature, filesize* типа uint) и *BitmapInfoHeader* (хранит поля *headerSize, width, height* типа uint), которые содержат в себе поля, соответствующие выбранной версии формата *bmp*. Ещё одна структура, созданная нами - *RGB* содержит три поля типа uchar, которые определяют один пиксель с помощью трёх цветов. Создадим

дополнительную структуру *image*, в которой будут определены поля с информацией о файле – *bmfh, bmih* и массив пикселей *rgb*.

2.3. Интерфейс командной строки

Строка, которой пользователь вводит команды имеет вид:
./src -[ключ команды] [имя обрабатываемого файла] -o [файл для сохранения] –[ключ1] [аргумент1] …

Для удобного пользования программой предусмотрено 10 ключей. Из них 5 ключей для обработки функций:

1). -i – для получения информации о файле;

2). -h – для вывода справки о файле;

3). -c – для выполнения обрезки файла;

4). -n – для инвертирования цветов в заданной окружности;

5). -s – для рисования отрезка.

Остальные 5 ключей являются дополнительными для ввода параметров. О них можно почитать в справке при запуске программы, либо в самом коде (см. приложение А)

Обработка запросов выполняется в функции *main*: создаётся строка с короткими ключей и сохраняется в переменную *opts* типа char, создаются объекты структуры *option*, в которых определяются длинные и короткие ключи, из библиотеки *getopt.h*. Если введённых аргументов меньше двух, то выводится сообщение об ошибке *“Wrong input, please enter keys to use the program.”* и вызывается функция помощи *help_output()*. Далее с помощью оператора *switch(opt)* перечисляются случаи обработки для каждого введённого ключа, в том числе обрабатываются возможные ошибки. Стандартное значение оператора *switch* установлено в виде вывода сообщения об ошибке: *“Wrong key”*. Если введённый ключ является ключом для функции обработки, то в этом *case* в переменную *funcName* типа *int* добавляется номер функции, чтобы вызывать её в отдельном операторе *switch(funcName)*. В этом операторе

значение по умолчанию является выводом сообщения об ошибке *“No function called”.*

2.4. Реализуемые функции

В программе содержатся две функции – считывание файла *readImage()* и *saveImage()*. Также была реализована дополнительная функция, которая проверяет файл на корректность и выводит ошибки-подсказки в обратном случае – *ifCorrect()*. Эта функция проверяет следующие случаи: находится ли bmp-файл в данной директории, сжат ли он, какая глубина кодирования и используется ли цветовая таблица.

Были реализованы следующие основные операции по обработке изображения:

1. Обрезка изображения

Данная операция осуществляется с помощью функции *void cut*. В неё передаются 6 аргументов: обрабатываемый файл (структуры *image)*, название файла, в котором нужно сохранить обработанный файл, 4 координаты типа int - *x1,y1,x2,y2*. Обработка возможных ошибок производится с помощью двух условий: первое – *x2* и *y2* не должны быть больше *x1* и *y1*, соответственно; второе – ни одна из координат не должна выходить за границы файла. Если данные введены корректно, то с помощью динамического выделения памяти создаётся новый файл, в который сохраняются только те пиксели, которые останутся внутри прямоугольника из заданных координат. Нужный нам прямоугольник получается с помощью вложенных циклов for. Новое изображение сохраняется с помощью функции *saveImage()*.

2. Инвертирование цвета в заданной окружности

Данная операция осуществляется с помощью функции *void paintOverTheCircle*. В неё передаётся 5 аргументов: обрабатываемый файл (структуры *image)*, название файла, в котором нужно сохранить обработанный файл, 3 координаты типа int - *x,y,R*. Обработка ошибок осуществляется подобным образом, как и в функции обрезки файла – с помощью двух условий.

Первое проверяет, не находятся ли переданные координаты центра за границами файла. Второе условие проверяет, не выходят ли за границы файла *(x+R)* и *(x-R)*. Если программа не вывела ошибку, то, проходя по вложенному циклу из двух for, инвертирует цвета компонентов *r,g,b* каждого пикселя, предварительно проверив, находятся ли эти пиксели внутри заданной окружности. Полученное изображение сохраняется с помощью функции *saveImage()*.

3. Рисование отрезка

Данная операция осуществляется с помощью одной основной функции *void drawSegment()* и двух побочных – *int accuracy()* и *int isLine()*. Следует начать с двух побочных: вторая проверяет, могут ли введённые координаты образовать линию, в этой же функции вызывается первая, которая проверяет, не превышают ли введённые аргументы заданную погрешность в виде половины ширины линии. Далее в основной функции обрабатываются ошибки: выход заданных координат начала и конца отрезка за границу файла, правильность введённых цветов (должны быть от 0 до 255), корректность толщины линии – она не должна быть меньше одного пикселя и не должна быть больше длины или ширины картинки. Если программа отработала не вывела ошибку, то она зайдёт во вложенный цикл из двух for и поменяет цвета пикселей в линии на заданный пользователем цвет. Обработанное изображение сохранится с помощью функции *saveImage()*.

Подробнее работу программы см. в приложении А.

3. ТЕСТИРОВАНИЕ

1. Если пользователь введёт ключ «-i», а затем название bmp-файла, то программа выведет информацию о файле (рис. 1)

```
rymarmary@laptop-rymarmary:/mnt/c/StudyProgs/coursework_images$ ./src -i moomintroll.bmp
signature:      4d42 (19778)
filesize:       7e936 (518454)
reserved1:      0 (0)
reserved2:      0 (0)
pixelArrOffset: 36 (54)
headerSize:     28 (40)
width:          1e0 (480)
height:         168 (360)
planes:         1 (1)
bitsPerPixel:   18 (24)
compression:    0 (0)
imageSize:      0 (0)
xPixelsPerMeter:        ec4 (3780)
yPixelsPerMeter:        ec4 (3780)
colorsInColorTable:     0 (0)
importantColorCount:    0 (0)
```

Рисунок 1 - Информация о файле

2. Если пользователь хочет получить справку по работе программы, то ему достаточно ввести один ключ «-h» (рис. 2)

```
rymarmary@laptop-rymarmary:/mnt/c/StudyProgs/coursework_images$ ./src -h
Hey, currently u're working with BMP Photo Editor 'Time for Edit'.
Here u can see a description of this program: it supports files of only 3rd version; encoding depth is 24 bits per color; file shouldn't be compressed.
Functions:
1). -i -- if u want to see an information about bmp-file
2). -h -- if u need to find out what this program can do (btw, u're reading it now)
3). -c -- if u want to cut file: eg, ./src -c korgi.bmp -o out.bmp -t 200,200,800,800
4). -n -- if u want to invert colors in a circle: eg, ./src -n korgi.bmp -o out.bmp -p 500,500 -r 30
5). -s -- if u want to draw a segment: eg, ./src -s korgi.bmp -o out.bmp -t 200,200,500,500 -b 0,185,0,30
Additional keys:
a). -o -- name of file, where u want to save a processed picture
b). -t -- two pairs of arguments, every number is divided from another by comma
c). -p -- one pair of arguments, every number is divided from another by comma
d). -r -- radius of the Circle u want to invert colors in
e). -b -- here u should enter 3 numbers of rgb color palette and bold of the segment, all numbers are divided by comma
Thanks for using 'Time for edit'.
```

Рисунок 2 - Вывод помощи

3. Если пользователь хочет обрезать изображение, то ему нужно ввести ключ «-c», потом название обрабатываемого файла, далее нужно написать ключ «-o» и название файла, в котором нужно сохранить обработанное изображение. Далее через ключ «-t» ввести координаты, в формате x1, y1, x2, y2. Если

программа отработала без ошибок, то будет выведено сообщение о том, что файл сохранён (рис. 3) Пример такой обработки изображения показан на рисунках 4 а и 4 б.

```
rymarmary@laptop-rymarmary:/mnt/c/StudyProgs/coursework_images$ ./src -c korgi.bmp -o out.bmp -t 100,100,600,600
The processed file is saved to out.bmp
```

Рисунок 3 - CLI при обрезке изображения

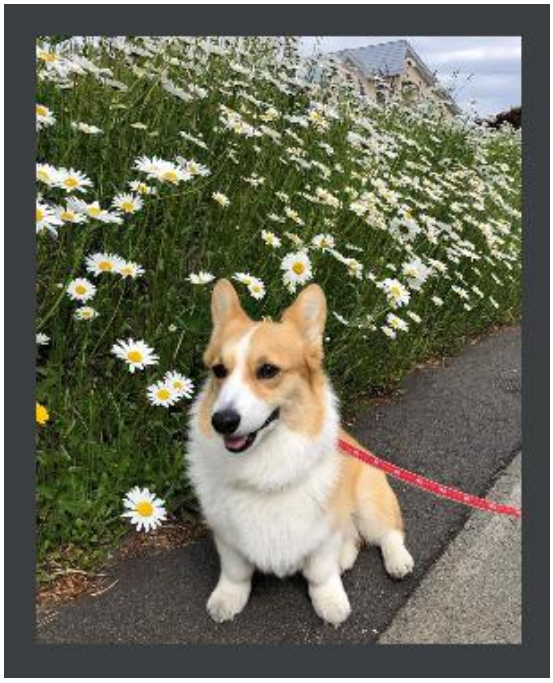
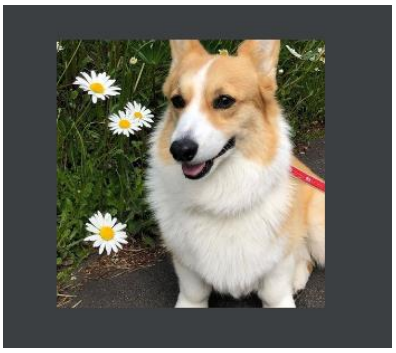
а)  б) 

Рисунок 4 – а). Изображение до обрезки; б). Изображение после обрезки

4. Если пользователь хочет инвертировать цвета в заданной окружности, то ему нужно ввести ключ «-n», потом название обрабатываемого файла, далее нужно написать ключ «-o» и название файла, в котором нужно сохранить обработанное изображение. Далее через ключ «-p» ввести координаты центра окружности, в формате x, y, а после этого радиус через ключ «-r». Если программа отработала без ошибок, то будет выведено сообщение о том, что файл сохранён (рис. 5) Пример такой обработки изображения показан на рисунках 6 а и 6 б.

```
rymarmary@laptop-rymarmary:/mnt/c/StudyProgs/coursework_images$ ./src -n moomintroll.bmp -o out1.bmp -p 100,100 -r 40
The processed file is saved to out1.bmp
```

Рисунок 5 – CLI при инвертировании цвета в заданной окружности

а)  б) 

Рисунок 6 – а). Изображение до инвертирования цвета в заданной окружности; б). Изображение после обработки

5. Если пользователь хочет нарисовать отрезок, то ему нужно ввести ключ «-s», потом название обрабатываемого файла, далее нужно написать ключ «-o» и название файла, в котором нужно сохранить обработанное изображение. Далее через ключ «-t» ввести координаты начала и конца отрезка, в формате x1, y1, x2, y2, а после этого через ключ «-b» ввести три цифры-компоненты rgb и толщину линии в пикселях. Если программа отработала без ошибок, то будет выведено сообщение о том, что файл сохранён (рис. 7) Пример такой обработки изображения показан на рисунках 8 а и 8 б.

```
rymarmary@laptop-rymarmary:/mnt/c/StudyProgs/coursework_images$ ./src -s catwithwine.bmp -o out2.bmp -t 10,10,750,750 -b 0,0,180,15
The processed file is saved to out2.bmp
```

Рисунок 7 – CLI при рисовании отрезка

а)  б) 

Рисунок 8 – а). Изображение до рисования отрезка; б). Изображение после обработки

6. Примеры обработки ошибок (рис. 9) : первая – был некорректно введён третий компонент цвета, вторая – была некорректно указана толщина линии, третья – был некорректно указан файл для обработки.

Рисунок 9 – Обработка ошибок

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы были изучены особенности работы с BMP файлами и написана программа, которая выполняет определённую обработку изображений: рисование отрезка, инвертирование цветов в заданной окружности и обрезка изображения. Для обработки запросов пользователя был реализован Command Line Interface с помощью библиотеки getopt.h. Чтобы программа не падала при некорректно введённых данных, в каждой функции осуществляются обработки возможных ошибок.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Б.В. Керниган. Язык программирования C : Учебник / Б.В. Керниган, Д.М. Ричи – М. : Изд-во Вильямс, 2019. – 277 с.

2. Статья в Wikipedia. URL: https://en.wikipedia.org/wiki/BMP_file_format (дата обращения: 13.07.2022)

3. Статья в Wikipedia. URL: https://en.wikipedia.org/wiki/Getopt (дата обращения: 15.07.2022)

4. Статья в Wikipedia. URL: https://en.wikipedia.org/wiki/C_standard_library (дата обращения: 19.07.2022)

5. Сайт онлайн-справочник. URL: http://www.c-cpp.ru (дата обращения: 27.07.2022)

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

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

#define length_file 256

#pragma pack(push, 1)
typedef struct {
    unsigned short signature;   // 0x4d42 | 0x4349 | 0x5450
    unsigned int filesize;      // размер файла
    unsigned short reserved1;   // 0
    unsigned short reserved2;   // 0
    unsigned  int  pixelArrOffset;//  смещение  до  поля  данных  (обычно
54=16+sizeOfStruct)
} BitmapFileHeader;

typedef struct {
    unsigned int headerSize;         // размер структуры
    unsigned int width;              // ширина в точках
    unsigned int height;             // высота в точках
    unsigned short planes;           // всегда должно быть 1
    unsigned short bitsPerPixel;     // 0 | 1 | 4 | 8 | 16 | 24 | 32
    unsigned int compression;        // BI_RGB | BI_RLE8 | BI_RLE4 |
                                     // BI_BITFIELDS | BI_JPEG | BI_PNG
                                     // реально используется лишь BI_RGB
    unsigned int imageSize;          // количество байт в поле данных
                                     // обычно устанавливается в 0
    unsigned int xPixelsPerMeter;    // горизонтальное разрешение
    unsigned int yPixelsPerMeter;    // вертикальное разрешение
    unsigned int colorsInColorTable; // кол-во используемых цветов
    unsigned int importantColorCount;// кол-во существенных цветов
} BitmapInfoHeader;

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

typedef struct {
    BitmapFileHeader bmfh;
    BitmapInfoHeader bmih;
    RGB **rgb;
} image;
#pragma pack(pop)

image readImage(char *name) {
    FILE *f = fopen(name, "rb");
    image img;
    fread(&img.bmfh, 1, sizeof(BitmapFileHeader), f);
    fread(&img.bmih, 1, sizeof(BitmapInfoHeader), f);
```

```
    unsigned int H = img.bmih.height;
    unsigned int W = img.bmih.width;

    img.rgb = malloc(H * sizeof(RGB *)+(4-(H*sizeof(RGB))%4)%4);
    for (int i = 0; i < H; i++) {
        img.rgb[i] = malloc(W * sizeof(RGB) + (4-(W*sizeof(RGB))%4)%4);
        fread(img.rgb[i], 1, W * sizeof(RGB) + (4-(W*sizeof(RGB))%4)%4,
f);
    }
    return img;
}

void saveImage(image *img, char *outName) {
    int length = (int) strlen(outName);
    if (outName[length - 1] != 'p' || outName[length - 2] != 'm' ||
outName[length - 3] != 'b' || outName[length - 4] != '.') {
        printf("Error: invalid file name, it must end with '.bmp'\n");
        return;
    }
    FILE *f = fopen(outName, "wb");
    if (!f) {
        printf("Error: unable to open the file\n");
        return;
    }
    fwrite(&img->bmfh, 1, sizeof(BitmapFileHeader), f);
    fwrite(&img->bmih, 1, sizeof(BitmapInfoHeader), f);

    unsigned int w = (img->bmih.width) * sizeof(RGB) + ((img->bmih.width
* sizeof(RGB)) * 3) % 4;
    for (int i = 0; i < img->bmih.height; i++) {
        fwrite(img->rgb[i], 1, w, f);
        free(img->rgb[i]);
    }
    printf("The processed file is saved to %s\n", outName);
    fclose(f);
}

void printFileInfo(image *img) {
    printf("signature:\t%x     (%hu)\n",     img->bmfh.signature,     img-
>bmfh.signature);
    printf("filesize:\t%x      (%u)\n",      img->bmfh.filesize,      img-
>bmfh.filesize);
    printf("reserved1:\t%x     (%hu)\n",     img->bmfh.reserved1,     img-
>bmfh.reserved1);
    printf("reserved2:\t%x     (%hu)\n",     img->bmfh.reserved2,     img-
>bmfh.reserved2);
    printf("pixelArrOffset:\t%x  (%u)\n",  img->bmfh.pixelArrOffset,  img-
>bmfh.pixelArrOffset);
    printf("headerSize:\t%x     (%u)\n",     img->bmih.headerSize,     img-
>bmih.headerSize);
    printf("width:     \t%x (%u)\n", img->bmih.width, img->bmih.width);
    printf("height:    \t%x (%u)\n", img->bmih.height, img->bmih.height);
    printf("planes:    \t%x (%hu)\n", img->bmih.planes, img->bmih.planes);
    printf("bitsPerPixel:\t%x   (%hu)\n",   img->bmih.bitsPerPixel,   img-
>bmih.bitsPerPixel);
    printf("compression:\t%x    (%u)\n",    img->bmih.compression,    img-
>bmih.compression);
```

```
    printf("imageSize:\t%x     (%u)\n",     img->bmih.imageSize,     img-
>bmih.imageSize);
    printf("xPixelsPerMeter:\t%x (%u)\n", img->bmih.xPixelsPerMeter, img-
>bmih.xPixelsPerMeter);
    printf("yPixelsPerMeter:\t%x (%u)\n", img->bmih.yPixelsPerMeter, img-
>bmih.yPixelsPerMeter);
    printf("colorsInColorTable:\t%x             (%u)\n",             img-
>bmih.colorsInColorTable, img->bmih.colorsInColorTable);
    printf("importantColorCount:\t%x            (%u)\n",             img-
>bmih.importantColorCount, img->bmih.importantColorCount);
}

int ifCorrect(image* img, char* name){
    unsigned long len = strlen(name);
    if (name[len - 1] != 'p' || name[len - 2] != 'm' || name[len - 3] !=
'b' || name[len - 4] != '.') {
        printf("Error: invalid file name, it must end with '.bmp'\n");
        return 1;
    }
    FILE* bmpfile = fopen(name, "rb");
    *img = readImage(name);
    if (!bmpfile){
        printf("There is not such file in the directory.\n");
        return 1;
    }
    if (img->bmih.compression != 0){
        printf("File is compressed, it shouldn't be.\n");
        return 1;
    }
    if (img->bmih.bitsPerPixel != 24){
        printf("File with such depth of encoding is not supported, it
should be 24 bits per pixel.\n");
        return 1;
    }
    if (img->bmih.colorsInColorTable != 0 || img->bmih.importantColorCount
!= 0){
        printf("File shouldn't use color table.\n");
        return 1;
    }
    return 0;
}

// cut function
void cut(image *img, char* nameOut, unsigned int x1, unsigned int y1,
unsigned int x2, unsigned int y2){
    if (x2<x1 || y2<y1){
        printf("Error: invailid arguments. There should be x2>x1 and
y2>y1.\n");
        return;
    }
    if (x1>img->bmih.height || x2>img->bmih.height || y1>img->bmih.width
|| y2>img->bmih.width || x1<1 || x2<1 || y1<1 || y2<1){
        printf("Error: enetered arguments are incorrect.\n");
        return;
    };
    image newImage;
    newImage.bmfh = img->bmfh;
```

```
    newImage.bmih = img->bmih;
    int W = x2-x1+1;
    int H = y2-y1+1;
    newImage.bmih.height = H;
    newImage.bmih.width = W;
    newImage.rgb = malloc(H*sizeof(RGB*));
    for (int i=0; i<H; i++){
        newImage.rgb[i] = malloc(W*sizeof(RGB)+(4-(W*sizeof(RGB))%4)%4);
    }
    for (int i=0; i<H; i++){
        for (int j=0; j<W; j++){
            newImage.rgb[i][j] = img->rgb[i+y1][j+x1];
        }
    }
    saveImage(&newImage, nameOut);
}

// paint over the circle function
void paintOverTheCircle(image *img, char* nameOut, unsigned int x, unsigned
int y, int R){
    if (x>img->bmih.height || y>img->bmih.width || x<1 || y<1){
        printf("Error:   enetered   arguments   of   coordinates   are
incorrect.\n");
        return;
    };
    if (x+R>img->bmih.height || y+R>img->bmih.width || x-R<0 || y-R<0){
        printf("Error: enetered argument of radius is incorrect.\n");
        return;
    };
    for (int i=x-R; i<x+R; i++){
        for (int j=y-R; j<y+R; j++){
            int radius = (x-j)*(x-j)+(y-i)*(y-i);
            if (radius<=(R*R)){
                img->rgb[i][j].r = 255 - img->rgb[i][j].r;
                img->rgb[i][j].g = 255 - img->rgb[i][j].g;
                img->rgb[i][j].b = 255 - img->rgb[i][j].b;
            }
        }
    }
    saveImage(img, nameOut);
}

// draw segment function
int accuracy(double arg1, double arg2, double accur){
    if (fabs(arg1-arg2) <= accur){
        return 1;
    }
    return 0;
}

int isLine(double x1, double y1, double x2, double y2, double i, double j,
double bold){
    if (x1 == x2){
        if ((j >= y1-bold/2 && j <= y2+bold/2) || (j >= y2-bold/2 && j<=
y2+bold/2)){
            if (accuracy(x1, i, 0.5*bold)){
                return 1;
```

```
            }
        }
    }
    double deltaX = x1 - x2;
    double deltaY = y1 - y2;
    double k = deltaY/deltaX;
    double b = (y1+y2-k*x1-k*x2) / 2;
    if (x1 < x2-bold/2 && x2 <= x1 && y2 >= y1){
        if (i >= x2 - bold / 2 && i <= x1 + bold / 2 && j >= y1 - bold /
2 && j <= y2 + bold / 2) {
            if (accuracy(j, k * i + b, 0.5 * bold))
                return 1;
            if (accuracy(i, (j - b) / k, 0.5 * bold))
                return 1;
        }
        if (i >= x1 - bold / 2 && j <= x2 + bold / 2 && j >= y2 - bold /
2 && j <= y1 + bold / 2) {
            if (accuracy(j, k * i + b, 0.5 * bold))
                return 1;
            if (accuracy(i, (j - b) / k, 0.5 * bold))
                return 1;
        }
    }
    if (x2 <= x1 && y2 <= y1 || x1 <= x2 && y1 <= y2) {
        if (i >= x2 - bold / 2 && i <= x1 + bold / 2 && j >= y2 - bold /
2 && j <= y1 + bold / 2) {
            if (accuracy(j, k * i + b, 0.5 * bold))
                return 1;
            if (accuracy(i, (j - b) / k, 0.5 * bold))
                return 1;
        }
        if (i >= x1 - bold / 2 && i <= x2 + bold / 2 && j >= y1 - bold /
2 && j <= y2 + bold / 2) {
            if (accuracy(j, k * i + b, 0.5 * bold))
                return 1;
            if (accuracy(i, (j - b) / k, 0.5 * bold))
                return 1;
        }
    }
    return 0;
}

void drawSegment(image *img, char* nameOut, unsigned int x1, unsigned int
y1, unsigned int x2,
                 unsigned int y2, int color1, int color2, int color3, int
bold){
    if (x1>img->bmih.height || x2>img->bmih.height || y1>img->bmih.width
|| y2>img->bmih.width || x1<1 || y1<1 || x2<1 || y2<1){
        printf("Error:   enetered   arguments   of   coordinates   are
incorrect.\n");
        return;
    };
    if (color1>255 || color1<0){
        printf("Error:   enetered   argument   of   the   1st   color   is
incorrect.\n");
        return;
    }
```

```
    if (color2>255 || color2<0){
        printf("Error:  enetered  argument  of  the  2nd  color  is
incorrect.\n");
        return;
    }
    if (color3>255 || color3<0){
        printf("Error:  enetered  argument  of  the  3rd  color  is
incorrect.\n");
        return;
    }
    if (bold<1 || bold>img->bmih.width || bold>img->bmih.height){
        printf("Error: entered argument of bold is incorrect.\n");
        return;
    }
    int H = img->bmih.height;
    int W = img->bmih.width;
    for (int i=0; i<H; ++i){
        for (int j=0; j<W; ++j){
            if (isLine(x1, y1, x2, y2, j, i, bold)){
                img->rgb[H-i-1][j].r = color1;
                img->rgb[H-i-1][j].g = color2;
                img->rgb[H-i-1][j].b = color3;
            }
        }
    }
    saveImage(img, nameOut);
}

// Command Line Interface

void help_output() {
    char info[] = "Hey, currently u're working with BMP Photo Editor 'Time
for Edit'.\n"
                  "Here  u  can  see  a  description  of  this  program:  it
supports files of only 3rd version; "
                  "encoding depth is 24 bits per color; file shouldn't be
compressed.\n"
                  "Functions:\n1). -i -- if u want to see an information
about bmp-file\n"
                  "2). -h -- if u need to find out what this program can
do (btw, u're reading it now)\n"
                  "3). -c -- if u want to cut file: eg, ./src -c korgi.bmp
-o out.bmp -t 200,200,800,800\n"
                  "4). -n -- if u want to invert colors in a circle: eg,
./src -n korgi.bmp -o out.bmp -p 500,500 -r 30\n"
                  "5). -s -- if u want to draw a segment: eg, ./src -s
korgi.bmp -o out.bmp -t 200,200,500,500 -b 0,185,0,30\n"
                  "Additional keys:\na). -o -- name of file, where u want
to save a processed picture\n"
                  "b).  -t  --  two pairs of arguments, every number is
divided from another by comma\n"
                  "c). -p -- one pair of arguments, every number is divided
from another by comma\n"
                  "d). -r -- radius of the Circle u want to invert colors
in\n"
                  "e). -b -- here u should enter 3 numbers of rgb color
palette and bold of the segment, all numbers are divided by comma\n"
```

```
                "Thanks for using 'Time for edit'.";
    puts(info);
}

int main(int argc, char *argv[]) {
    char *opts = "hi:c:t:s:n:o:p:b:r:"; //если без аргументов, то без
двоеточия
    struct option longOpts[] = {{"help", no_argument, NULL, 'h'},
                                {"info", required_argument, NULL, 'i'},
                                {"cut", required_argument, NULL, 'c'},
                                {"segment",   required_argument,   NULL,
's'},
                                {"negate", required_argument, NULL, 'n'},
                                {"outputFile",  required_argument,  NULL,
'o'},
                                {"twoPairOfCoordinates",
required_argument, NULL, 't'},
                                {"onePairOfCoordinates",
required_argument, NULL, 'p'},
                                {"radiusCircle", required_argument, NULL,
'r'},
                                {"colorsBold",  required_argument,  NULL,
'b'},
                                {NULL, 0, NULL}};
    int opt;
    int longOpt;
    opt = getopt_long(argc, argv, opts, longOpts, &longOpt);
    image img;
    char inputFile[length_file];
    char outputFile[length_file];
    int numArgs;
    int x1Coordinate=0, y1Coordinate=0, x2Coordinate=0, y2Coordinate=0;
    int color1=0, color2=0, color3=0, bold=0;
    int radius=-1;
    int funcName=0;

    if (argc<2){
        printf("Wrong input, please enter keys to use the program.\n");
        help_output();
        return 0;
    }

    while (opt != -1){
        switch (opt) {
            case 'h':{
                help_output();
                return 0;
            }
            case 'i':{
                sscanf(optarg, "%s", inputFile);
                img = readImage(inputFile);
                if (ifCorrect(&img, inputFile) != 0){
                    printf("Incorrect input file, please try another.\n");
                    return 1;
                }
                printFileInfo(&img);
                return 0;
```

```
            }
            case 'c':{
                numArgs = sscanf(optarg, "%s", inputFile);
                if (numArgs<1){
                    printf("Too few arguments.\n");
                    return 1;
                }
                if (ifCorrect(&img, inputFile) != 0){
                    printf("Incorrect input file, please try another.\n");
                    return 1;
                }
                funcName = 1;
                break;
            }
            case 'n':{
                numArgs = sscanf(optarg, "%s", inputFile);
                if (numArgs<1){
                    printf("Too few arguments.\n");
                    return 1;
                }
                if (ifCorrect(&img, inputFile) != 0){
                    printf("Incorrect input file, please try another.\n");
                    return 1;
                }
                funcName = 2;
                break;
            }
            case 's':{
                numArgs = sscanf(optarg, "%s", inputFile);
                if (numArgs<1){
                    printf("Too few arguments.\n");
                    return 1;
                }
                if (ifCorrect(&img, inputFile) != 0){
                    printf("Incorrect input file, please try another.\n");
                    return 1;
                }
                funcName = 3;
                break;
            }
            case 't':{
                numArgs = sscanf(optarg, "%d,%d,%d,%d", &x1Coordinate,
&y1Coordinate, &x2Coordinate, &y2Coordinate);
                if (numArgs<4){
                    printf("Too few arguments for two pair of
coordinates.\n");
                    return 1;
                }
                break;
            }
            case 'p':{
                numArgs = sscanf(optarg, "%d,%d", &x1Coordinate,
&y1Coordinate);
                if (numArgs<2){
                    printf("Too few arguments for one pair of
coordinates.\n");
                    return 1;
```

```
                }
                break;
            }
            case 'r':{
                numArgs = sscanf(optarg, "%d", &radius);
                if (numArgs<1){
                    printf("You haven't entered radius.\n");
                    return 1;
                }
                break;
            }
            case 'b':{
                numArgs = sscanf(optarg, "%d,%d,%d,%d", &color1, &color2,
&color3, &bold);
                if (numArgs<4){
                    printf("Too few arguments for colors and bold. There
should be 3 arguments for colors and 1 for bold.\n");
                    return 1;
                }
                break;
            }
            case 'o':{
                sscanf(optarg, "%s", outputFile);
                break;
            }
            default:{
                printf("Wrong key\n");
                return 1;
            }
        }
        opt = getopt_long(argc, argv, opts, longOpts, &longOpt);
    }
    switch (funcName){
        case 1: {
            cut(&img,     outputFile,     x1Coordinate,     y1Coordinate,
x2Coordinate, y2Coordinate);
            break;
        }
        case 2:{
            if (radius==-1) {
                radius = abs((x2Coordinate - x1Coordinate) / 2);
                x1Coordinate = (x2Coordinate + x1Coordinate) / 2;
                x2Coordinate = (y2Coordinate + y1Coordinate) / 2;
            }
            paintOverTheCircle(&img,      outputFile,      x1Coordinate,
y1Coordinate, radius);
            break;
        }
        case 3:{
            drawSegment(&img,  outputFile,  x1Coordinate,  y1Coordinate,
x2Coordinate, y2Coordinate, color1, color2, color3, bold);
            break;
        }
        default:{
            printf("No function called.\n");
            break;
        }
```

```
    }
    return 0;
}
```