

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

КУРСОВАЯ РАБОТА
подисциплине «Программирование»
Тема: Работа с BMP-файлами

Студент(ка) гр.0381

Березовская В.В.

Преподаватель

Берленко Т. А.

Санкт-Петербург
2021

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студентка Березовска В.В.

Группа 0381

Тема работы: Работа с BMP-файлами

Исходные данные:

Дан BMP файл.

Надо реализовать рисование окружности, копирование области, отражение области.

Содержание пояснительной записи:

Аннотация, содержание, введение, описание архитектуры программы, вывод результата, примеры работы.

Предполагаемый объем пояснительной записи:

Не менее 15 страниц.

Дата выдачи задания: 05.04.2021

Дата сдачи курсовой: 25.05.2021

Дата защиты курсовой: 27.05.2021

Студентка гр.0381

Березовская В.В.

Преподаватель

Берленко Т. А.

АННОТАЦИЯ

В данной курсовой работе реализованы функции по работе с изображениями в формате BMP, а именно: рисование окружности, копирование заданной области, отражение области. Всё это предоставляется конечному пользователю в виде консольного приложения. Для написания CLI был использован getopt. Разработка велась на языке С.

В результате было разработано консольное приложение, запрашивающее у пользователя файл формата BMP, после чего предоставляющее ему возможность всячески изменять файл, а полученный результат сохранить.

SUMMARY

In this course work, functions with images in BMP format are implemented, namely: drawing a circle, copying an area, reflecting an area. All this is presented to the end user in the form of an application with a command-line interface. The Getopt framework was used to write the CLI. The development was carried out in the C language. As a result, an application was developed that prompts the user for a BMP file.

СОДЕРЖАНИЕ

1. Аннотация.....	3
2. Summary	3
3. Введение	5
4.Краткое описание кода.....	6
4.1 СТРУКТУРЫ, ФУНКЦИИ И ИХ ОБРАБОТКИ.....	6
4.2 ФУНКЦИИ ДЛЯ РЕАЛИЗАЦИИ ЗАДАНИЙ ПОЛЬЗОВАТЕЛЯ.....	7
4.3 ВСПОМОГАТЕЛЬНЫЕ И ДРУГИЕ ФУНКЦИИ.....	9
5.Инструкция по использованию.....	11
6.Проверка работы программы.....	12
7.Заключение.....	14
8.Приложение А.....	14

Введение

Цель работы:

Научиться работать файлами в формате BMP, реализовать функционал, заданный в условии.

Задачи работы:

Для достижения поставленной цели требуется решить следующие задачи:

1. Считывание изображение в структуры BitmapFileHeader, BitmapInfoHeader и Rgb (в динамическую память). Программа работает только с картинками формата BMP, в которых используется 24 цвета.
2. Реализация интерфейса в командной строке в виде CLI для запроса действий от пользователя.
3. Реализация подзадач (решение каждой подзадачи вынесено в отдельную функцию):
 - Рисование окружности двумя методами: по заданным координатам центра окружности и длины её радиуса; по заданным координатам левого верхнего угла и правого нижнего угла квадрата, в который она вписана.
 - Отражение заданной координатами левого верхнего угла и правого нижнего угла области относительно оси на выбор (x или y), то есть по горизонтали и вертикали.
 - Копирование заданной координатами левого верхнего угла и правого нижнего угла области-источника в левый верхний угол заданной области-назначения.

4.1. СТРУКТУРЫ, ФУНКЦИИ И ИХ ОБРАБОТКИ

1. *struct BitmapFileHeader;* - Структура, которая хранит отметку для отличия формата от других (сигнатуру формата), размер файла в байтах, положение пиксельных данных относительно начала данной структуры (в байтах).
2. *struct BitmapInfoHeader;* - Структура, которая хранит ширину, высоту и битность раstra, а также формат пикселей, информацию о цветовой таблице и разрешении.
3. *struct Rgb;* - Структура, которая хранит в себе пиксели заданного изображения в формате red, green, blue, значения которых от 0 до 255.
4. *Void printFileHeader(BitmapFileHeader header);* - Функция, которая выводит на экран информацию, хранящуюся в *struct BitmapFileHeader*
5. *void printInfoHeader(BitmapInfoHeader header);* - Функция, которая выводит на экран информацию, хранящуюся в *struct BitmapInfoHeader*
6. *struct option longOpts[] = {};* -Структура, хранящая в себе длинные названия команд и их короткие ключи для CLI, своеобразный словарь

4.2. ФУНКЦИИ ДЛЯ РЕАЛИЗАЦИИ ЗАДАНИЙ ПОЛЬЗОВАТЕЛЯ

1. *void Reflection (Rgb **arr, BitmapFileHeader bmfh, BitmapInfoHeader bmif, unsigned int H, unsigned int W, int xleft, int xright, int yleft, int yright, int axis);* - функция, которая в зависимости от оси отражения, бежит по всем координатам изображения и меняет их местами с помощью вспомогательной функции *swap_st*. Например, при отражении по горизонтали(x) мы бежим от правого нижнего угла выделенной области до левого верхнего угла по координатам x и поочерёдно меняем местами. Аналогично делаем с отражением по вертикали(y), только цикл пускаем до середины картинки и перебираем значения y. По окончанию работы функция записывает изменения в файл **out.bmp**.
2. *void Copying (Rgb **arr, BitmapFileHeader bmfh, BitmapInfoHeader bmif, unsigned int H, unsigned int W, int xleft, int xright, int yleft, int yright, int xplace, int yplace)* - функция, которая отвечает за реализацию копирования заданной координатами левого верхнего угла и правого нижнего угла области-источника в левый верхний угол заданной области-назначения. На вход получает массив пикселей, все данные об изображении, координаты левого верхнего угла и правого нижнего угла области-источника и координаты левого верхнего угла области-назначения. С помощью цикла for происходит запись пикселей области-источника из массива пикселей **arr** в новый массив **Rgb **copy**. Далее происходит замена пикселей в массиве arr на пиксели в массиве copy начиная с позиции области-назначения. В конце функция записывает результат работы с изображением в файл **out.bmp**.
3. *void Square_circle (Rgb **arr, BitmapFileHeader bmfh, BitmapInfoHeader bmif, unsigned int H, unsigned int W, int xleft, int xright, int yleft, int yright, int count, int red, int green, int blue, int variation, int red_fill, int green_fill, int blue_fill)* - функция, реализующая построения окружности по заданным координатам левого верхнего угла и правого нижнего угла квадрата, в который она вписана. На вход получает массив пикселей, все данные об изображении, координаты левого верхнего угла и правого нижнего угла квадрата, толщину окантовки и цвет, значение (0 или 1), в качестве аргумента отсутствия/наличия заливки

окружности, цвет заливки. Простейшими математическими действиями вычисляются координаты центра окружности и её радиус, при помощи функции ***void circle*** происходит рисование окружности. При значении variation==1 вызывается функция ***void circle_fill***, которая осуществляет заливку окружности заданного цвета. В конце функция записывает результат работы с изображением в файл **out.bmp**.

4. ***void circle_radius (Rgb **arr, BitmapFileHeader bmfh, BitmapInfoHeader bmif, unsigned int H, unsigned int W, int xcenter,int ycenter,int r, int count, int variation, int red, int green, int blue, int red_fill, int green_fill, int blue_fill)*** - функция, которая отвечает за построение окружности по заданным координатам центра окружности и длины её радиуса. На вход получает всё тоже самое, что и предыдущая функция, только вместо координат углов квадрата, координаты центра окружности и длину радиуса. Остальной алгоритм полностью повторяет ***Square_circle*** и в конце функция записывает результат работы с изображением в файл **out.bmp**.

4.3. ВСПОМОГАТЕЛЬНЫЕ И ДРУГИЕ ФУНКЦИИ

1. **void printHelp();**- вспомогательная функция, которая осуществляет вывод справки на экран.
2. **int isNum(char *n) ;** - вспомогательная функция, которая осуществляет проверку условия. Если входная строка является числом – возвращает 1, иначе 0.
3. **int Defence(int a, int b, int c);** - вспомогательная функция, которая осуществляет проверку условия. Возвращает 1, если аргументы a и b положительные, а меньше или равно b, а также b меньше или равно c, иначе возвращает 0.
4. **int checkLeftRight(int xleft, int xright, int yleft, int yright, int H, int W);** - вспомогательная функция, которая осуществляет проверку условия на основании вспомогательной функции **int Positive(int a, int b, int c)** (подробнее в пункте 3.4). Возвращает 1, если значение функции **Defence()** для координат по оси x и для координат по оси y == 1, иначе возвращает 0.
5. **int checkSquare(int xleft, int xright, int yleft, int yright) {};**- вспомогательная функция, которая осуществляет проверку условия. Возвращает 1, если координаты образуют квадрат, иначе возвращает 0.
6. **void circle (Rgb **arr, BitmapFileHeader bmfh, BitmapInfoHeader bmif, unsigned int H, unsigned int W, int xleft, int xright, int yleft, int yright, int red, int green, int blue, int r, int count, int xcenter, int ycenter);** – функция, которая осуществляет рисование окружности. На вход получает массив пикселей, все данные об изображении, координаты левого верхнего угла и правого нижнего угла квадрата, цвет окантовки, длина радиуса, толщина окантовки, координаты центра окружности. Функция перебирает все пиксели изображения и окрашивает те, которые удовлетворяют математическому условию окружности $x^2 + y^2 = r^2$. В конце функция записывает результат работы с изображением в файл out.bmp.

7. **void circle_fill (Rgb **arr, BitmapFileHeader bmfh, BitmapInfoHeader bmif, unsigned int H, unsigned int W, int xleft, int xright, int yleft, int yright, int red, int green, int blue, int r, int xcenter, int ycenter) ;** -
функция, которая осуществляет заливку окружности. На вход получает массив пикселей, все данные об изображении, координаты левого верхнего угла и правого нижнего угла квадрата, цвет заливки, длина радиуса, и координаты центра окружности. Функция зацикливает перебор всех пикселей изображения и окрашивает те, которые удовлетворяют математическому условию окружности $x^2 + y^2 = r^2$, постепенно уменьшает радиус, пока он не станет ==0. В конце функция записывает результат работы с изображением в файл out.bmp.
8. **int checkColor (int a);** -вспомогательная функция, которая осуществляет проверку условия. Возвращает 1, если чисто находится в диапазоне от 0 до 255, иначе возвращает 0.

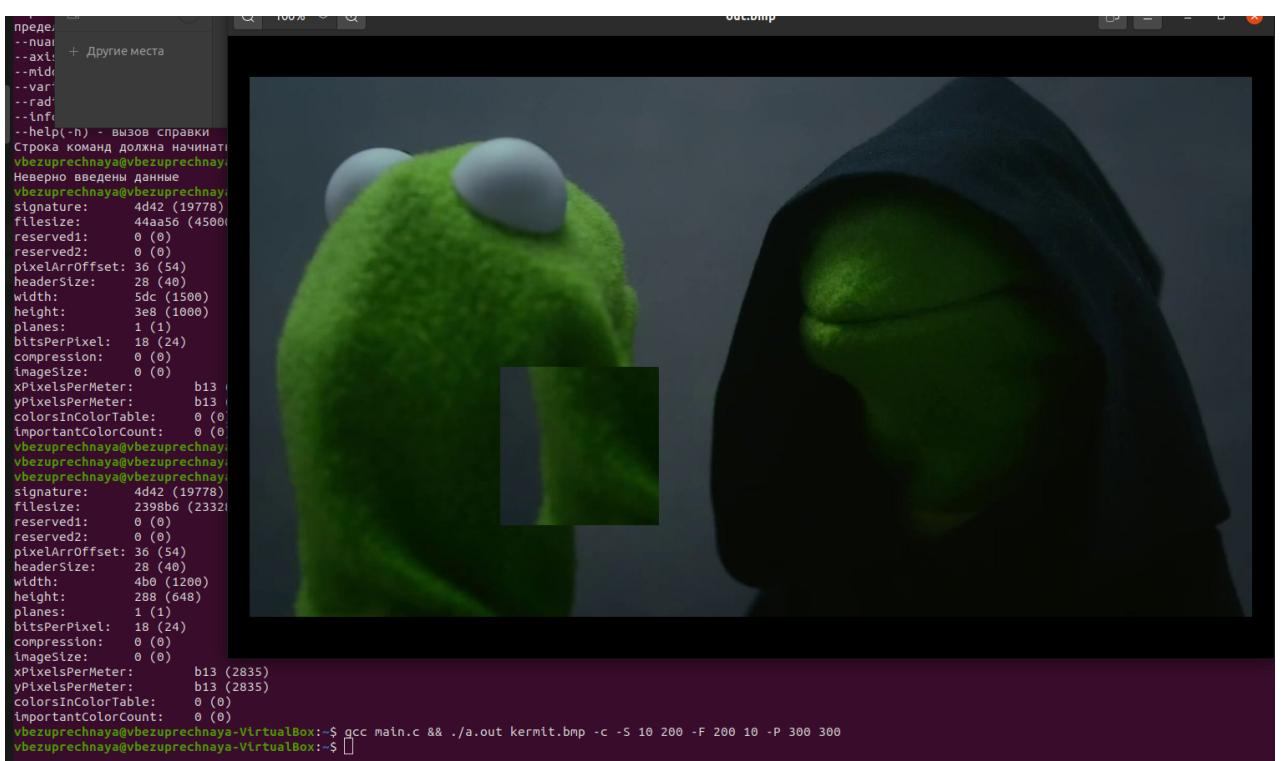
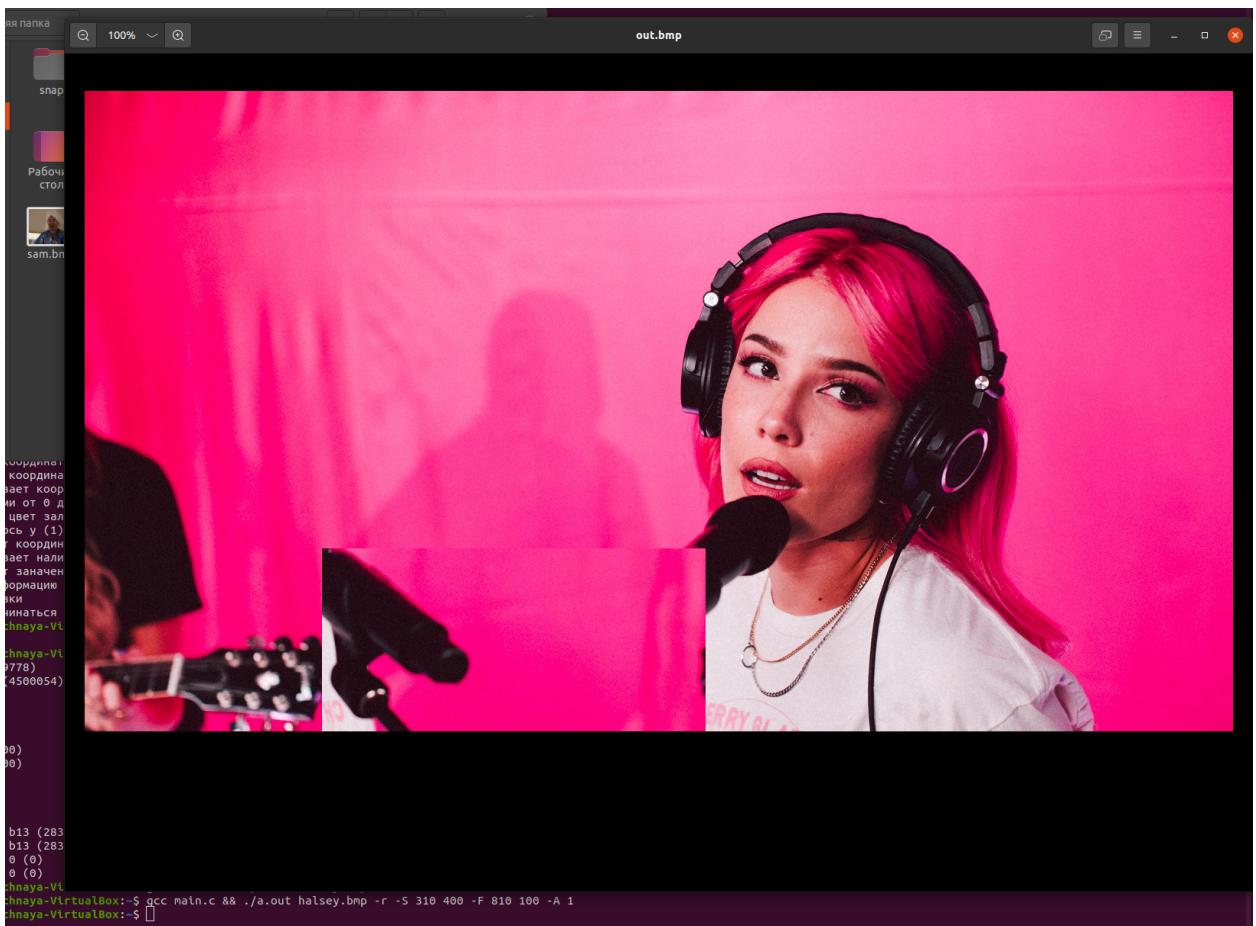
5. ИНСТРУКЦИЯ ПО ИСПОЛЬЗОВАНИЮ

При запуске программы необходимо ввести название_файла.bmp и следующие команды, как и в обычной утилит-консоли Linux.

- reflect(-r) - Отражение заданной области по выбранной оси. Координаты задается при помощи -S и -F. Так же выбирается ось, по которой происходит отражение -A;
- copying(-c) - Копирование заданной области. Участок задается при помощи -S и -F и координат левого верхнего угла области-назначения -P;
- disc_radius(-d) - Рисование окружности. Окружность определяется при помощи координат ее центра -M и радиуса -R. Толщина и цвет окантовки окружности выбираются при помощи -C (цвет) и -T (толщина). Заливка окружности (да(1)/нет(0)) и её цвет определяются при помощи -V (наличие --square_circle(-s) - Рисование окружности. Окружность определяется при помощи -S и -F. Толщина и цвет окантовки окружности выбираются при помощи -C (цвет) и -T (толщина). Заливка окружности (да(1)/нет(0)) и её цвет --start(-S) - считывает координаты (целый числа) верхнего левого угла прямоугольной области (-S 10 100);
- finish(-F) - считывает координаты (целый числа) нижнего правого угла прямоугольной области (-F 100 10);
- placement(-P) - считывает координаты (целый числа) левого верхнего угла области-назначения (-P 200 200);
- thickness(-T) - считывание толщины окантовки (-T 4) ;
- color(-C) - считывает цвет окантовки. Цвет определяется тремя числами от 0 до 255. (-C 230 5 10);
- nuance(-N) - считывает цвет заливки. Цвет определяется тремя числами от 0 до 255. (-N 0 255 10) ;
- axis(-A) - считывает ось у (1) или ось x (2) (-A 1) ;
- middle(-M) - считывает координаты (целые числа) центра окружности (-M 100 100) ;
- variation(-V) - считывает наличие (1) или отсутствие (0) заливки. (-V 1) ;
- radius(-R) - считывает значение радиуса (-R 40) ;
- info(-i) - выводит информацию о входном файле ;

--help(-h) - вызов справки ;

Примеры работы программы:



```

width: + Другие места
height: + Другие места
planes: 0 (0)
bitsPerPixel: 0 (0)
compression: 0 (0)
image:
<Pixel>
xPixelsPerMeter: b13 (2835)
yPixelsPerMeter: b13 (2835)
colorsInColorTable: 0 (0)
importantColorCount: 0 (0)
/vbezuprechnaya@vbezuprechnaya-VirtualBox:~$ gcc main.c && ./a.out hal
/vbezuprechnaya@vbezuprechnaya-VirtualBox:~$ gcc main.c && ./a.out hal
/vbezuprechnaya@vbezuprechnaya-VirtualBox:~$ gcc main.c && ./a.out ker
signature: 4d42 (19778)
filesize: 2398b6 (2332854)
reserved1: 0 (0)
reserved2: 0 (0)
pixelArrOffset: 36 (54)
headerSize: 28 (40)
width: 4b0 (1200)
height: 288 (648)
planes: 1 (1)
bitsPerPixel: 18 (24)
compression: 0 (0)
imageSize: 0 (0)
xPixelsPerMeter: b13 (2835)
yPixelsPerMeter: b13 (2835)
colorsInColorTable: 0 (0)
importantColorCount: 0 (0)
/vbezuprechnaya@vbezuprechnaya-VirtualBox:~$ gcc main.c && ./a.out ker
/vbezuprechnaya@vbezuprechnaya-VirtualBox:~$ gcc main.c && ./a.out sam
signature: 4d42 (19778)
filesize: c0192 (786834)
reserved1: 0 (0)
reserved2: 0 (0)
pixelArrOffset: 36 (54)
headerSize: 28 (40)
width: 233 (563)
height: 1d1 (465)
planes: 1 (1)
bitsPerPixel: 18 (24)
compression: 0 (0)
imageSize: 0 (0)
xPixelsPerMeter: b13 (2835)
yPixelsPerMeter: b13 (2835)
colorsInColorTable: 0 (0)
importantColorCount: 0 (0)
/vbezuprechnaya@vbezuprechnaya-VirtualBox:~$ gcc main.c && ./a.out sam.bmp -d -M 200 200 -R 50 -C 255 0 255 -T 3 -V 1 -N 0 0 0
/vbezuprechnaya@vbezuprechnaya-VirtualBox:~$ 

```

```

/vbezuprechnaya@vbezuprechnaya-VirtualBox:~$ gcc main.c && ./a.out ker
/vbezuprechnaya@vbezuprechnaya-VirtualBox:~$ gcc main.c && ./a.out
signature: 4d42 (19778)
filesize: c0192 (786834)
reserved1: 0 (0)
reserved2: 0 (0)
pixelArrOffset: 36 (54)
headerSize: 28 (40)
width: 233 (563)
height: 1d1 (465)
planes: 1 (1)
bitsPerPixel: 18 (24)
compression: 0 (0)
imageSize: 0 (0)
xPixelsPerMeter: b13 (2835)
yPixelsPerMeter: b13 (2835)
colorsInColorTable: 0 (0)
importantColorCount: 0 (0)
/vbezuprechnaya@vbezuprechnaya-VirtualBox:~$ gcc main.c && ./a.out
/vbezuprechnaya@vbezuprechnaya-VirtualBox:~$ gcc main.c && ./a.out
signature: 4d42 (19778)
filesize: 2bf236 (2880054)
reserved1: 0 (0)
reserved2: 0 (0)
pixelArrOffset: 36 (54)
headerSize: 28 (40)
width: 4b0 (1200)
height: 320 (800)
planes: 1 (1)
bitsPerPixel: 18 (24)
compression: 0 (0)
imageSize: 0 (0)
xPixelsPerMeter: b13 (2835)
yPixelsPerMeter: b13 (2835)
colorsInColorTable: 0 (0)
importantColorCount: 0 (0)
/vbezuprechnaya@vbezuprechnaya-VirtualBox:~$ gcc main.c && ./a.out stich.bmp -s -S 600 600 -F 900 300 -C 220 0 205 -T 3 -V 1 -N 0 46 0
/vbezuprechnaya@vbezuprechnaya-VirtualBox:~$ 

```

ЗАКЛЮЧЕНИЕ

В результате работы было реализовано считывание BMP-файла в структуры и динамическую память, созданы различные функции для обработки пикселей, интерфейс в виде CLI для взаимодействия с пользователем и получена программа, работающая на Linux.

Приложение А

Название файла: Berezovskaia_Valeriya_cw.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <locale.h>
#include <getopt.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 printFileHeader(BitmapFileHeader header) {
```

```

        printf("signature:\t%x (%hu)\n", header.signature,
header.signature);
        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("headerSize:\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);
}

void swap(char *a, char *b) {
    char t = *a;
    *a = *b;
    *b = t;
}

void swap_st(Rgb* a, Rgb* b) {
    Rgb t = *a;
    *a= *b;
    *b= t;
}

int isNum(char *n) {
    for(int i = 0; i < strlen(n); i++) {

```

```

        if (isdigit(n[i]))
            continue;
        else
            return 0;
    }

    return 1;
}

int Defence(int a, int b, int c) {
    if ( (a>=0) && (b>=0) && (a<=b) && (b<c) ) {
        return 1;
    }
    else {
        return 0;
    }
}

int checkLeftRight(int xleft, int xright, int yleft, int yright, int H,
int W) {
    if (Defence(xleft, xright, W) && Defence(yright, yleft, H)) {
        return 1;
    }
    else {
        return 0;
    }
}

int checkSquare(int xleft, int xright, int yleft, int yright) {
    if ((xright-xleft)==(yleft-yright)) {
        return 1;
    }
    else {
        return 0;
    }
}

void Reflection (Rgb **arr, BitmapFileHeader bmfh, BitmapInfoHeader bmif,
unsigned int H, unsigned int W, int xleft, int xright, int yleft, int
yright, int axis) {
    int count=0;
    switch (axis) {
        case 1:
            for (int i=yright; i<yleft; i++) {
                for (int count=0; count<((xright-xleft)/2);
count++) {
                    swap_st(&arr[i][xleft+count],

```

```

&arr[i][xright-count]);
}
}
break;
case 2:
    while (count<(yleft-yright)/2) {
        for (int j=xleft; j<xright; j++) {
            swap_st(&arr[yright+count][j],
&arr[yleft-count][j]);
        }
        count++;
    }
    break;
default:
    printf("Введите, пожалуйста, одну из предложенных
цифр\n");
}
FILE *ff = fopen("out.bmp", "wb");

bmif.height = H;
bmif.width = W;
fwrite(&bmfh, 1, sizeof(BitmapFileHeader), ff);
fwrite(&bmif, 1, sizeof(BitmapInfoHeader), ff);
unsigned int wi = (W) * sizeof(Rgb) + (W)%4;
for(int i=0; i<H; i++) {
    fwrite(arr[i], 1, wi, ff);
}
fclose(ff);

}

void Copying (Rgb **arr, BitmapFileHeader bmfh, BitmapInfoHeader bmif,
unsigned int H, unsigned int W, int xleft, int xright, int yleft, int
yright, int xplace, int yplace) {
    Rgb **copy = malloc((yleft-yright)*sizeof(Rgb*));
    for (int i=0; i<(yleft-yright); i++) {
        copy[i] = malloc((xright-xleft) * sizeof(Rgb) +
(xright-xleft)%4);
        for (int j=0; j<(xright-xleft); j++) {
            copy[i][j]=arr[yright+i][xleft+j];
        }
    }
    yplace-= (yleft-yright);

    for (int i=yplace; i<(yplace+(yleft-yright)); i++) {
        for (int j=xplace; j<(xplace+(xright-xleft)); j++) {
            arr[i][j]=copy[i-yplace][j-xplace];
        }
    }
}

```

```

        }

    }

FILE *ff = fopen("out.bmp", "wb");
bmif.height = H;
bmif.width = W;
fwrite(&bmfh, 1, sizeof(BitmapFileHeader), ff);
fwrite(&bmif, 1, sizeof(BitmapInfoHeader), ff);
unsigned int wi = (W) * sizeof(Rgb) + (W)%4;
for(int i=0; i<H; i++) {
    fwrite(arr[i], 1, wi, ff);
}
fclose(ff);
free(copy);

}

void circle (Rgb **arr, BitmapFileHeader bmfh, BitmapInfoHeader bmif,
unsigned int H, unsigned int W, int xleft, int xright, int yleft, int
yright, int red, int green, int blue, int r, int count, int xcenter, int
ycenter) {
    for (int i=0; i<=count; i++) {
        for (int y=yright; y<=yleft; y++) {
            for (int x=xleft; x<=xright; x++) {
                if ( ((xcenter-x)*(xcenter-x)+(ycenter-y)*(ycenter-y))<=(r*r) &&
((xcenter-x)*(xcenter-x)+(ycenter-y)*(ycenter-y))>=((r-1)*(r-1)) ) {
                    arr[y][x].r=red;
                    arr[y][x].g=green;
                    arr[y][x].b=blue;
                }
            }
        }
        r++;
   yleft++;
    yright--;
    xleft--;
    xright++;
    }
}

FILE *ff = fopen("out.bmp", "wb");
bmif.height = H;
bmif.width = W;
fwrite(&bmfh, 1, sizeof(BitmapFileHeader), ff);
fwrite(&bmif, 1, sizeof(BitmapInfoHeader), ff);
unsigned int wi = (W) * sizeof(Rgb) + (W)%4;
for(int i=0; i<H; i++) {

```

```

        fwrite(arr[i],1,wi,ff);
    }
    fclose(ff);
}

void circle_fill (Rgb **arr, BitmapFileHeader bmfh, BitmapInfoHeader bmif,
unsigned int H, unsigned int W, int xleft, int xright, int yleft, int
yright, int red, int green, int blue, int r, int xcenter, int ycenter) {
    while (r>=0) {
        for (int y=yright; y<=yleft; y++) {
            for (int x=xleft; x<=xright; x++) {
                if
(((xcenter-x)*(xcenter-x)+(ycenter-y)*(ycenter-y))<(r*r)) {
                    arr[y][x].r=red;
                    arr[y][x].g=green;
                    arr[y][x].b=blue;
                }
            }
        }
        r--;
    }

FILE *ff = fopen("out.bmp", "wb");
bmif.height = H;
bmif.width = W;
fwrite(&bmfh, 1, sizeof(BitmapFileHeader), ff);
fwrite(&bmif, 1, sizeof(BitmapInfoHeader), ff);
unsigned int wi = (W) * sizeof(Rgb) + (W)%4;
for(int i=0; i<H; i++) {
    fwrite(arr[i],1,wi,ff);
}
fclose(ff);

}

int checkColor (int a) {
    if (a>=0 && a<=255) {
        return 1;
    }
    else {
        return 0;
    }
}

void Square_circle (Rgb **arr, BitmapFileHeader bmfh, BitmapInfoHeader
bmif, unsigned int H, unsigned int W, int xleft, int xright, int yleft,

```

```

int yright, int count, int red, int green, int blue, int variation, int
red_fill, int green_fill, int blue_fill) {
    int r = (xright-xleft)/2;
    int r_remember=r;
    int ycenter=(yleft-r);
    int xcenter = xright-r;
    circle(arr, bmfh, bmif, H, W, xleft, xright, yleft, yright, red,
green, blue, r, count, xcenter, ycenter);
    switch (variation) {
        case 1:
            r_remember--;
            circle_fill(arr, bmfh, bmif, H, W, xleft, xright, yleft,
yright, red_fill, green_fill, blue_fill, r_remember, xcenter, ycenter);
            break;
        case 0:
            break;
        default:
            printf("Введите, пожалуйста, одну из предложенных
цифр\n");
    }
}

```

```

void circle_radius (Rgb **arr, BitmapFileHeader bmfh, BitmapInfoHeader
bmif, unsigned int H, unsigned int W, int xcenter,int ycenter,int r, int
count, int variation, int red, int green, int blue, int red_fill, int
green_fill, int blue_fill) {
    int xleft, xright, yleft, yright;
    xright=xcenter+r;
    xleft=xcenter-r;
    yleft=ycenter+r;
    yright=ycenter-r;
    int r_remember=r;
    circle(arr, bmfh, bmif, H, W, xleft, xright, yleft, yright, red,
green, blue, r, count, xcenter, ycenter);
    switch (variation) {
        case 1:
            r_remember--;
            circle_fill(arr, bmfh, bmif, H, W, xleft, xright, yleft,
yright, red_fill, green_fill, blue_fill, r_remember, xcenter, ycenter);
            break;
        case 0:
            break;
        default:
            printf("Введите, пожалуйста, одну из предложенных
цифр\n");
    }
}

```

```

void printHelp()
{
    printf("Справка\n\n");
    printf("--reflect(-r) - Отражение заданной области по выбранной оси.  

Координаты задается при помощи -S и -F. Так же выбирается ось, по которой  

происходит отражение -A\n");
    printf("(Пример 1) file_name.bmp --reflect -S 10 100 -F 100 10 -A  

1\n");
    printf("(Пример 2) file_name.bmp -r -S 10 100 -F 100 10 -A 2\n\n");
    printf("--copying(-c) - Копирование заданной области. Участок задается  

при помощи -S и -F и координат левого верхнего угла области-назначения  

-P\n");
    printf("(Пример) file_name.bmp --copying -S 10 200 -F 200 10 -P 300  

300\n\n");
    printf("--disc_radius(-d) - Рисование окружности. Окружность  

определяется при помощи координат ее центра -M и радиуса -R. Толщина и  

цвет окантовки окружности выбираются при помощи -C (цвет) и -T (толщина).  

Заливка окружности (да(1)/нет(0)) и её цвет определяются при помощи -V  

(наличие заливки) и -N (цвет)\n");
    printf("(Пример 1) file_name.bmp --disc_radius -M 50 150 -R 20 -C 255  

0 0 -T 4 -V 0\n");
    printf("(Пример 2) file_name.bmp -d -M 50 150 -R 20 -C 255 0 0 -T 6 -V  

1 -N 0 255 255\n\n");
    printf("--square_circle(-s) - Рисование окружности. Окружность  

определяется при помощи -S и -F. Толщина и цвет окантовки окружности  

выбираются при помощи -C (цвет) и -T (толщина). Заливка окружности  

(да(1)/нет(0)) и её цвет определяются при помощи -V (наличие заливки) и  

-N (цвет)\n");
    printf("(Пример 1) file_name.bmp --square_circle -S 10 100 -F 100 10  

-C 255 0 0 -T 4 -V 0\n");
    printf("(Пример 2) file_name.bmp -s -S 10 100 -F 100 10 -C 255 0 0 -T  

6 -V 1 -N 0 200 255\n\n");
    printf("--start(-S) - считывает координаты (целый числа) верхнего  

левого угла прямоугольной области (-S 10 100)\n");
    printf("--finish(-F) - считывает координаты (целый числа) нижнего  

правого угла прямоугольной области (-F 100 10)\n");
    printf("--placement(-P) - считывает координаты (целый числа) левого  

верхнего угла области-назначения (-P 200 200)");
    printf("--thickness(-T) - считывание толщины окантовки (-T 4)");
    printf("--color(-C) - считывает цвет окантовки. Цвет определяется тремя  

числами от 0 до 255. (-C 230 5 10)\n");
    printf("--nuance(-N) - считывает цвет заливки. Цвет определяется тремя  

числами от 0 до 255. (-N 0 255 10)\n");
    printf("--axis(-A) - считывает ось у (1) или ось x (2) (-A 1)\n");
    printf("--middle(-M) - считывает координаты (целые числа) центра  

окружности (-M 100 100)\n");
    printf("--variation(-V) - считывает наличие (1) или отсутствие (0)  

заливки. (-V 1)\n");
}

```

```

printf("--radius(-R) - считывает значение радиуса (-R 40)\n");
printf("--info(-i) - выводит информацию о входном файле\n");
printf("--help(-h) - вызов справки\n");
printf("Строка команд должна начинаться названием файла ввода\n");

}

int main(int argc, char **argv){

    int xleft=0, xright=0, yleft=0, yright=0, xcenter=0, ycenter=0, r=0,
xplace=0, yplace=0;
    int red=0, green=0, blue=0, red_fill=0, green_fill=0, blue_fill=0;
    char* file_name=(char*)calloc(100, sizeof(char));
    int count=0, axis=0, variation=0;
    unsigned int H;
    unsigned int W;
    char *optstring = "rcdsS:F:P:T:C:N:A:M:V:R:ih";
    int flag=0, opt=0;
    int longIndex=0;

struct option longOpts[] = {
    {"copying", no_argument, &flag, 'c'},
    {"reflect", no_argument, &flag, 'r'},
    {"disc_radius", no_argument, &flag, 'd'},
    {"square_circle", no_argument, &flag, 's'},
    {"help", no_argument, &flag, 'h'},
    {"info",no_argument, &flag, 'i'},
    {"variation", required_argument, NULL, 'V'},
    {"thickness", required_argument, NULL, 'T'},
    {"start", required_argument, NULL, 'S'},
    {"finish", required_argument, NULL, 'F'},
    {"color", required_argument, NULL, 'C'},
    {"nuance", required_argument, NULL, 'N'},
    {"axis", required_argument, NULL, 'A'},
    {"placement", required_argument, NULL, 'P'},
    {"middle", required_argument, NULL, 'M'},
    {"radius", required_argument, NULL, 'R'},
    {NULL, 0, NULL, 0}
};

FILE *f = fopen(argv[1], "rb");
if (!(f)){
    printf("Невозможно открыть файл\n");
    return 0;
}
BitmapFileHeader bmfh;
BitmapInfoHeader bmif;

```

```

fread(&bmfh,1,sizeof(BitmapFileHeader),f);
fread(&bmif,1,sizeof(BitmapInfoHeader),f);

H = bmif.height;
W = bmif.width;

Rgb **arr = malloc(H*sizeof(Rgb*));
for(int i=0; i<H; i++) {
    arr[i] = malloc(W * sizeof(Rgb) + W%4);
    fread(arr[i],1,W * sizeof(Rgb) + W%4,f);
}
if (!(f)) {
    printf("Невозможно открыть файл\n");
    return 0;
}
opt = getopt_long(argc, argv, optstring, longOpts, &longIndex);
while (opt!= -1) {
    switch (opt) {
        case 'S':
            if (!isNum(optarg)) {
                printf("%s - не целое число\n", optarg);
                return 0;
            }

            if (!isNum(argv[optind])) {
                printf("%s - не целое число\n", argv[optind]);
                return 0;
            }

            xleft = atoi(optarg);
            yleft = atoi(argv[optind]);
            break;
        case 'F':
            if (!isNum(optarg)) {
                printf("%s - не целое число\n", optarg);
                return 0;
            }

            if (!isNum(argv[optind])) {
                printf("%s - не целое число\n", argv[optind]);
                return 0;
            }

            xright = atoi(optarg);
            yright = atoi(argv[optind]);
            break;
    }
}

```

```

case 'R':
    if (!isNum(optarg)) {
        printf("%s - это не целое число\n", optarg);
        return 0;
    }
    r = atoi(optarg);
    break;
case 'M':
    if (!isNum(optarg)) {
        printf("%s - не целое число\n", optarg);
        return 0;
    }
    if (!isNum(argv[optind])) {
        printf("%s - не целое число\n", argv[optind]);
        return 0;
    }
    xcenter = atoi(optarg);
    ycenter = atoi(argv[optind]);
    break;

case 'C':
    if (!isNum(optarg)) {
        printf("%s - не целое число\n", optarg);
        return 0;
    }
    if (!isNum(argv[optind])) {
        printf("%s - не целое число\n", argv[optind]);
        return 0;
    }
    if (!isNum(argv[optind+1])) {
        printf("%s - не целое число\n", argv[optind]);
        return 0;
    }
    red=atoi(optarg);
    green=atoi(argv[optind]);
    blue=atoi(argv[optind+1]);
    break;
case 'N':
    if (!isNum(optarg)) {
        printf("%s - не целое число\n", optarg);
        return 0;
    }
    if (!isNum(argv[optind])) {
        printf("%s - не целое число\n", argv[optind]);
        return 0;
    }
}

```

```

    if (!isNum(argv[optind+1])){
        printf("%s - не целое число\n", argv[optind]);
        return 0;
    }
    red_fill=atoi(optarg);
    green_fill=atoi(argv[optind]);
    blue_fill=atoi(argv[optind+1]);
    break;

case 'A':
    if (!strcmp("1", optarg) || !strcmp("2", optarg))
        axis = atoi(optarg);
    else{
        printf("Неверно указан метод построения
пентаграммы (возможные значения: 1 и 2)\n");
        return 0;
    }
    break;
case 'P':
    if (!isNum(optarg)){
        printf("%s - не целое число\n", optarg);
        return 0;
    }
    if (!isNum(argv[optind])){
        printf("%s - не целое число\n", argv[optind]);
        return 0;
    }
    xplace = atoi(optarg);
    yplace = atoi(argv[optind]);
    break;
case 'T':
    if(!isNum(optarg)) {
        printf("%s - это не целое число\n", optarg);
        return 0;
    }
    count = atoi(optarg);
    count--;
    break;
case 'V':
    if (!strcmp("1", optarg) || !strcmp("0", optarg))
        variation = atoi(optarg);
    else{
        printf("Неверно указан метод построения
пентаграммы (возможные значения: 0 и 1)\n");
        return 0;
    }
}

```

```

        break;
    case 'c':
        flag = 'c';
        break;
    case 'r':
        flag = 'r';
        break;
    case 'd':
        flag = 'd';
        break;

    case 's':
        flag = 's';
        break;

    case 'i':
        flag = 'i';
        break;

    case 'h':
        flag = 'h';
        break;
    }

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

switch (flag) {
    case 'r':
        if (!(checkLeftRight(xleft, xright, yleft, yright, H, W))) {
            printf("Неверно введены данные\n");
            return 0;
        }
        else {
            Reflection(arr, bmfh, bmif, H, W, xleft, xright, yleft,
yright, axis);
        }
        break;
    case 'c':
        if (!(checkLeftRight(xleft, xright, yleft, yright, H, W)) ||
((!(checkLeftRight(xplace, xplace, yplace, yplace, H, W))) ||
(! (checkLeftRight((xplace+(xright-xleft)), (xplace+(xright-xleft)),
(yplace-(yleft-yright)), (yplace-(yleft-yright)), H, W))))) {
            printf("Неверно введены данные\n");
            return 0;
        }
        else {

```

```

        Copying(arr, bmfh, bmif, H, W, xleft, xright, yleft,
yright, xplace, yplace);
    }
    break;
case 'd':
    xright=xcenter+r;
    xleft=xcenter-r;
    yleft=ycenter+r;
    yright=ycenter-r;
    if (!(checkLeftRight(xleft, xright, yleft, yright, H, W)) ||
(xright+count>W) || (xleft-count<0) || (yleft+count>H) || (yright-count<0)
|| (count<=-1)) || ( !(checkColor(red))) || !(checkColor(green))) ||
!(checkColor(blue))) ) || ( !(checkColor(red_fill))) ||
!(checkColor(green_fill))) || !(checkColor(blue_fill))) ) ) {
        printf("Неверно введены данные\n");
        return 0;
    }
else {
    circle_radius(arr, bmfh, bmif, H, W, xcenter, ycenter, r,
count, variation, red, green, blue, red_fill, green_fill, blue_fill);
}
break;
case 's':
    if (!(checkLeftRight(xleft, xright, yleft, yright, H, W)) ||
(xright+count>W) || (xleft-count<0) || (yleft+count>H) || (yright-count<0)
|| (count<=-1)) || ( !(checkColor(red))) || !(checkColor(green))) ||
!(checkColor(blue))) ) || ( !(checkColor(red_fill))) ||
!(checkColor(green_fill))) || !(checkColor(blue_fill))) ) ) {
        printf("Неверно введены данные\n");
        return 0;
    }
else {
    Square_circle(arr, bmfh, bmif, H, W, xleft, xright, yleft,
yright, count, red, green, blue, variation, red_fill, green_fill,
blue_fill);
}
break;
case 'i':
    printFileHeader(bmfh);
    printInfoHeader(bmif);
    break;
case 'h':
    printHelp();
    break;
} free (arr);
return 0;
}

```