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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: обработка изображений на языке Си

Студент гр. 0382 _____ Кондратов Ю.А

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2021

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Кондратов Ю.А

Группа 0382

Тема работы: обработка изображений на языке Си

Программа должна реализовывать весь следующий функционал по обработке bmp-файла

Рисование правильного шестиугольника. Шестиугольник определяется: либо координатами левого верхнего и правого нижнего угла квадрата, в который он вписан, либо координатами его центра и радиусом в который он вписан, толщиной линий, цветом линий, шестиугольник может быть залит или нет, цветом которым залит шестиугольник, если пользователем выбран залитый.

Копирование заданной области. Функционал определяется:

Координатами левого верхнего угла области-источника

Координатами правого нижнего угла области-источника

Координатами левого верхнего угла области-назначения

Заменяет все пиксели одного заданного цвета на другой цвет. Функционал определяется:

Цвет, который требуется заменить

Цвет на который требуется заменить

Сделать рамку в виде узора. Рамка определяется:

Узором (должно быть несколько на выбор. Красивый узор можно получить используя фракталы)

Цветом

Шириной

Содержание пояснительной записки:

Отчет должен содержать подробное описание выполненной вами работы, программной реализации того или иного функционала.

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания:

Дата сдачи работы:

Дата защиты работы:

Студент _____ Кондратов Ю.А.

Преподаватель _____ Берленко Т.А.

АННОТАЦИЯ

В процессе выполнения курсовой работы была реализована программа для обработки изображений на языке Си. Обработка изображения производилась при помощи функций стандартной библиотеки языка Си и специально написанных функций. Для удобства использования программа была реализована в виде консольной утилиты.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Данная курсовая работа предназначена для отработки навыков программирования и применения теоретических знаний и практических умений для решения такой прикладной задачи, как обработка изображения.

В качестве объекта изучения выбран BMP формат хранения графических файлов. В этом формате изображение представляется в виде двумерного массива пикселей и в таком же виде происходит обработка изображения.

Для запуска программы необходимо сначала вызвать в терминале утилиту make при помощи команды make.

1. ЦЕЛЬ И ЗАДАЧИ

Цель: разработка стабильной программы, способной выполнять обработку изображения в соответствие с командами пользователя.

Для достижения цели необходимо выполнить следующие задачи:

- реализовать считывание BMP файла;
- реализовать все необходимые для обработки файла функции;
- предоставить пользователю возможность подачи команд программе (с помощью функций из из заголовочного файла getopt.h).

2. ХОД ВЫПОЛНЕНИЯ РАБОТЫ

2.1 Считывание изображения

Файлы в формате BMP состоят из заголовка файла (file header), информационного заголовка (info header), непосредственно массива пикселей прочих опциональных частей. Для хранения заголовков были реализованы структуры соответственно BitmapFileHeader и BitmapInfoHeader. Массив пикселей состоит из структур Rgb, в которых скомпанованы значения красной, зелёной и синей компонент пикселя. Для хранения непосредственно BMP файла реализована структура BmpFile. Считывание файла происходит при помощи функции readBmp, в этой функции последовательно считываются заголовки, а затем и сам массив пикселей.

2.2 Функции обработки изображения

1) Функция рисования правильного шестиугольника drowHexagon.

Реализация данной функции основана на следующем алгоритме: сначала вычисляются координаты вершин шестиугольника,

```
int x[6], y[6];
for (int i = 0; i < 6; i++) {
    double angle = M_PI / 180 * (60 * i - 30);
    x[i] = x_c + r * cos(angle);
    y[i] = y_c + r * sin(angle);
}
```

далее эти точки соединяются линиями при помощи функции drawLine, основанной на алгоритме Брезенхема. Необходимая толщина линий достигается при помощи дорисовывания нескольких таких же линий внутрь шестиугольника (для того чтобы не менялся внешний радиус, заданный пользователем). Заливка шестиугольника реализуется при помощи дорисовывания некоторого количества шестиугольников внутри исходного. Если пользователь задаёт шестиугольник при помощи координат двух вершин квадрата, в который должен быть вписан шестиугольник, то по этим

координатам вычисляются координаты центра и радиус, по которым уже рисуется шестиугольник по описанному выше алгоритму.

2) Функция копирования заданной области copyArea.

Данная функция работает следующим образом: заданная пользователем область копируется, в двумерный массив, после чего данные из этого массива копируются в область назначения.

3) Функция замены цвета changeColor.

В данной функции в цикле при помощи функции rgbCmp сравниваются цвет текущего пикселя с цветом, который нужно заменить. Если результат положительный, то цвет этого пикселя изменяется.

4) Функции создания рамок в виде узора.

4.1) Функция создания градиентной рамки gradientFrame.

В данной функции в цикле рисуется некоторое количество рамок ширины 1 (это количество равно ширине рамки, заданной пользователем), причём в каждой рамке значение каждой компоненты цвета, заданного пользователем в качестве начального, постепенно изменяется в сторону значений компонент цвета, заданного пользователем в качестве конечного.

4.2) Функция создания рамки с узором «в полоску».

В данной функции рисуется количество рамок ширины один равное ширине рамки, заданной пользователем, причём в зависимости от чётности номер рамки (считается от края изображения) рамка имеет либо первый цвет заданный пользователем, либо второй.

4.3) Функция рисования волнистой рамки.

В данной функции узор рамки задаётся при помощи функции |a*sin(bx)|, где параметры a и b зависят от ширины рамки.

```
for (int y = 0; y < thickness; y++) {
      int f_x = (int) fabs(thickness * sin(2.0 / thickness * x));
      if (y <= f_x)
            bf->pixel_array[y][x] = color;
}
```

2.3 Запись изображения

Запись изображения в файл производится при помощи функции writeBmp. Эта функция сначала записывает в файл информацию из заголовков (File Header и Info Header), а затем и сам массив пикселей.

2.3 Создание интерфейса

Считывание пользовательских данный производится при помощи аргументов функции main (int argc, char *argv[]) и функции getopt_long().

Сначала программой обрабатываются исключительные ситуации: когда передан только один аргумент (название программы), ты выводится справка по утилите, если произошла ошибка при открытия файла на считывание или если не введено название опции, то производится выход из программы. Далее название опции (оно передаётся не в виде ключа, а в виде обычной строки) записывается в переменную choice. Далее создаются переменные, необходимые для хранения всех ключей, после чего с помощью функции getAllKeys в эти переменные (если передан соответствующий ключ) записываются значения.

Алгоритм работы функции getAllKeys основан на использовании функции getopt_long(). Для корректной работы этой функции создан специальный массив структур option long_opts[] и строка short_opts. Далее в цикле while рассматриваются все ключи, которые сортируются при помощи конструкции switch.

5. ТЕСТИРОВАНИЕ

1) Тестирование опции c_hexagon.

Пользовательский ввод: ./bed sample.bmp c_hexagon -z 300.300 -r 200 -c 200.200.200 -t 20 -f -C 142.28.70 -o output.bmp

Содержимое output.bmp:

Вывод: программа работает верно.

2) Тестирование опции s_hexagon.

Пользовательский ввод: ./bed sample.bmp s_hexagon -s 700.700 -e 300.300 -c 17.140.80 -t 20 -o output.bmp

Содержимое output.bmp:

Вывод: программа работает верно.

3) Тестирование функции copy_area.

Пользовательский ввод: ./bed sample.bmp copy_area -s 700.700 -e 300.300 -z 1000.1000 -o output.bmp

Содержимое output.bmp:

Вывод: программа работает верно.

4) Тестирование функции pattern_frame.

Пользовательский ввод: ./bed sample.bmp pattern_frame -p 1 -c 0.0.255 -C 255.0.255 -t 100

Содержимое output.bmp:

Вывод: программа работает верно.

4. ЗАКЛЮЧЕНИЕ

В процессе курсовой работы была выполнена цель: разработана стабильная программа, способная выполнять обработку изображения в формате BMP в соответствии с командами пользователя.

Для выполнения цели были решены следующие задачи:

- реализовано считывание BMP файла;
- реализованы все необходимые для обработки файла функции;
- пользователю предоставлена возможность подачи команд программе (с помощью функций из из заголовочного файла getopt.h).

6. СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. BMP file format URL: https://en.wikipedia.org/wiki/BMP_file_format

2. Керниган Б. и Ритчи Д. Язык программирования Си. М.: Вильямс, 1978 288 с.

3. Cplusplus URL: http://cplusplus.com

4.Алгоритм Брезенхема URL: https://ru.wikipedia.org/wiki/Алгоритм_Брезенхэма

6. CLI URL: https://www.gnu.org/software/libc/manual

ПРИЛОЖЕНИЕ А

Исходный код программы

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

#pragma pack (push, 1)
typedef struct {
    uint8_t b;
    uint8_t g;
    uint8_t r;
} Rgb;

typedef struct {
    uint16_t signature;
    uint32_t filesize;
    uint16_t reserved1;
    uint16_t reserved2;
    uint32_t pixelArrOffset;
} BitmapFileHeader;

typedef struct {
    uint32_t headerSize;
    uint32_t width;
    uint32_t height;
    uint16_t planes;
    uint16_t bitsPerPixel;
    uint32_t compression;
    uint32_t imageSize;
    uint32_t xPixelsPerMeter;
    uint32_t yPixelsPerMeter;
    uint32_t colorsInColorTable;
    uint32_t importantColorCount;
```

```
} BitmapInfoHeader;

#pragma pack(pop)

typedef struct {
    BitmapFileHeader bfh;
    BitmapInfoHeader bih;
    Rgb **pixel_array;
} BmpFile;

int readBmp(char *file_name, BmpFile *bmpf);

void writeBmp(char *file_name, BmpFile *bmpf);

int rgbCmp(Rgb *c1, Rgb *c2);

void changeColor(Rgb ref_color, Rgb color_to_change, BmpFile *bf);

void copyArea(int x0, int y0, int x1, int y1, int x_dest, int y_dest,
BmpFile *bf);

void drawLine(int x0, int y0, int x1, int y1, Rgb color, BmpFile *bf);

void drawHexagon(int x_c, int y_c, int r, Rgb color, int thickness, int
fill, Rgb fill_color, BmpFile *bf);

void gradientFrame(Rgb start_color, Rgb end_color, int frame_width,
BmpFile *bf);

void stripedFrame(Rgb color1, Rgb color2, int frame_width, BmpFile *bf);

void wavyFrame(Rgb color, int thickness, BmpFile *bf);

void getAllKeys(int argc, char *argv[], int p1[][2], int p2[][2], int
p3[][2], int *rad, int *thickness, int *fill_flag, int *pattern, Rgb
*color1, Rgb *color2, char **wfile);

void printHelp() {
```

```
    printf("\nBed - это утилита для редактирование графических файлов в формате \".bmp\".\n");
    printf("Формат поддерживаемых файлов:\n");
    printf("\t* BMP version 3;\n");
    printf("\t* без таблицы цветов;\n");
    printf("\t* глубина цвета - 24 бита;\n");
    printf("\t* без сжатия.\n");
    printf("Формат вызова программы: ./bed [путь к исходному файлу] [название опции] [ключи и аргументы]\n");
    printf("Функции программы:\n");
    printf("\t- Рисование правильного шестиугольника:\n");
    printf("\t\t1) По координатам квадрата - s_hexagon.\n\t\t2) По координиатам центра и радиусу - c_hexagon.\n");
    printf("\t- Копирование выбранной области - copy_area.\n");
    printf("\t- Замена цвета - change_color.\n");
    printf("\t- Рамка в виде узора - pattern_frame.\n");
    printf("\t- Вывод информации о файле - info.\n");
    printf("Ключи:\n \t-s/--start [координата 1].[координата 2] - верхний левый угол квадрата/копируемой области;\n");
    printf("\t-e/--end [координата 1].[координата 2] - нижний правый угол квадрата/копируемой области;\n");
    printf("\t-z/--centre [координата 1].[координата 2] - центр окружности, в которую вписан шестиугольник;\n");
    printf("\t-r/--radius [значение] - радиус этой окружности;\n");
    printf("\t-t/--thickness [значение] - толщина контура шестиугольника/ширина рамки;\n");
    printf("\t-f/--fill - заливка шестиугольника;\n");
    printf("\t-p/--pattern [значение] - выбор узора рамки (1 - градиент, 2 - узор \"в полосочку\", 3 - волнистый узор (1 и 2 задаются двумя цветами));\n");
    printf("\t-c/--color1 [значение красной компоненты].[значение зелёной компоненты].[значение синей компоненты] - цвет границы шестиугольника/заменяемый цвет/начальный цвет градиента/цвет волнистой рамки (по умолчанию чёрный);\n");
    printf("\t-C/--color2 [значение красной компоненты].[значение зелёной компоненты].[значение синей компоненты] - цвет заливки шестиугольника/заменяющий цвет/конечный цвет градиента (по умолчанию чёрный);\n");
```

```
    printf("\t-o/--output [название файла] - название файла, в который
требуется записать изменённый файл (по умолчанию - это название исходного
файла)\n");
}

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);
}

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

    if (argc == 1 || !strcmp(argv[1], "help")) {
```

```
        printHelp();
        return 0;
    }

    BmpFile bmpf;
    if (!readBmp(argv[1], &bmpf)) return 0;

    if (argc < 3) {printf("Не введено название опции.\n"); return 0;}
    char *choice = argv[2];

    int p1[2] = {-1, -1}, p2[2] = {-1, -1}, p3[2] = {-1, -1};
    int rad = -1, thick = -1, fill_flag = 0, pattern = -1;
    Rgb color1 = {0, 0, 0}, color2 = {0, 0, 0};
    char *wfile_name = argv[1];

    opterr = 0;
    getAllKeys(argc, argv, &p1, &p2, &p3, &rad, &thick, &fill_flag,
&pattern, &color1, &color2, &wfile_name);

    if (!strcmp(choice, "c_hexagon")){
        if (p3[0] == -1 || p3[1] == -1 || p3[0] >= bmpf.bih.width ||
p3[1] >= bmpf.bih.height)
        {printf("Центр шестиугльника не может находиться вне картинки.\
n"); return 0;}
        if (rad == -1) {printf("Неправильно введеён радиус.\n"); return
0;}
        if (thick < 0 || thick > rad/2) {printf("Неправильно введена
толщина линии.\n"); return 0;}
        drawHexagon(p3[0], p3[1], rad, color1, thick, fill_flag, color2,
&bmpf);
    }

    else if (!strcmp(choice, "s_hexagon")){
        if (p1[0] > p2[0]) {int tmp = p1[0]; p1[0] = p2[0]; p2[0] = tmp;}
        if (p1[1] > p2[1]) {int tmp = p1[1]; p1[1] = p2[1]; p2[1] = tmp;}

        if (p1[1] - p1[0] != p2[1] - p2[0]) { printf("Координаты не
образуют квадрат.\n"); return 0;}
```

```
        p3[0] = p1[0] + (p2[0] - p1[0]) / 2;
        p3[1] = p1[1] + (p2[1] - p1[1]) / 2;
        rad = (p2[0] - p1[0]) / 2;

        if (p3[0] == -1 || p3[1] == -1 || p3[0] >= bmpf.bih.width ||
p3[1] >= bmpf.bih.height)
        {printf("Неправильно введены координаты центра.\n"); return 0;}
        if (rad == -1) {printf("Неправильно введеён радиус.\n"); return
0;}
        if (thick < 0 || thick > rad/2) {printf("Неправильно введена
толщина линии.\n"); return 0;}
        drawHexagon(p3[0], p3[1], rad, color1, thick, fill_flag, color2,
&bmpf);
    }

    else if (!strcmp(choice, "copy_area")){
        if (!(p1[0] >= 0 && p1[0] < bmpf.bih.width &&
              p1[1] >= 0 && p1[1] < bmpf.bih.height &&
              p2[0] >= 0 && p2[0] < bmpf.bih.width &&
              p2[1] >= 0 && p2[1] < bmpf.bih.height &&
              p3[0] >= 0 && p3[0] < bmpf.bih.width &&
              p3[1] >= 0 && p3[1] < bmpf.bih.height))
        {printf("Неверно заданы координаты точек.\n"); return 0;}
        copyArea(p1[0], p1[1], p2[0], p2[1], p3[0], p3[1], &bmpf);
    }

    else if (!strcmp(choice, "change_color")){
        changeColor(color2, color1, &bmpf);
    }
    else if (!strcmp(choice, "pattern_frame")){
        if (thick > bmpf.bih.height/3 || thick > bmpf.bih.width/3)
{printf("Недопустимая ширина рамки.\n"); return 0;}
        switch (pattern){
            case 1:
                gradientFrame(color1, color2, thick, &bmpf);
                break;
            case 2:
                stripedFrame(color1, color2, thick, &bmpf);
                break;
```

```
            case 3:
                wavyFrame(color1, thick, &bmpf);
                break;
            default:
                printf("Неверно задан номер узора.\n");
        }
    }
    else if (!strcmp(choice, "info")){
        printFileHeader(bmpf.bfh);
        printInfoHeader(bmpf.bih);
    }

    else printf("Неизвестное название опции.\n");

    writeBmp(wfile_name, &bmpf);
    return 0;
}

void getAllKeys(int argc, char *argv[], int p1[][2], int p2[][2], int
p3[][2], int *rad, int *thickness, int *fill_flag, int *pattern, Rgb
*color1, Rgb *color2, char **wfile){
    struct option long_opts[] = {
            {"start", required_argument, NULL, 's'},
            {"end", required_argument, NULL, 'e'},
            {"centre", required_argument, NULL, 'z'},
            {"radius", required_argument, NULL, 'r'},
            {"thickness", required_argument, NULL, 't'},
            {"fill", no_argument, NULL, 'f'},
            {"pattern", required_argument, NULL, 'p'},
            {"color1", required_argument, NULL, 'c'},
            {"color2", required_argument, NULL, 'C'},
            {"output", required_argument, NULL, 'o'},
            {NULL, 0, NULL, 0}
    };
    char *short_opts = "s:r:z:e:r:t:fp:c:C:o:";
    int opt;

    while ((opt = getopt_long(argc, argv, short_opts, long_opts, NULL)) !
= -1){
```

```
switch (opt){
    case 's':{
        int ind = optind - 1;
        int arg_len = strlen(argv[ind]);

        if (!isdigit(argv[ind][0])) break;
        (*p1)[0] = atoi(argv[ind]);

        int i = 0;
        for (; argv[ind][i] != '.'; i++)
            if (i >= arg_len) break;
        if (i == arg_len) break;

        if (!isdigit(argv[ind][i + 1])) break;
        (*p1)[1] = atoi(&argv[ind][i + 1]);

        break;
    }
    case 'e':{
        int ind = optind - 1;
        int arg_len = strlen(argv[ind]);

        if (!isdigit(argv[ind][0])) break;
        (*p2)[0] = atoi(argv[ind]);

        int i = 0;
        for (; argv[ind][i] != '.'; i++)
            if (i >= arg_len) break;
        if (i == arg_len) break;

        if (!isdigit(argv[ind][i + 1])) break;
        (*p2)[1] = atoi(&argv[ind][i + 1]);

        break;
    }
    case 'z':{
        int ind = optind - 1;
        int arg_len = strlen(argv[ind]);
```

```
    if (!isdigit(argv[ind][0])) break;
    (*p3)[0] = atoi(argv[ind]);

    int i = 0;
    for (; argv[ind][i] != '.'; i++)
        if (i >= arg_len) break;
    if (i == arg_len) break;

    if (!isdigit(argv[ind][i + 1])) break;
    (*p3)[1] = atoi(&argv[ind][i + 1]);

    break;
}
case 'r':{
    if (!isdigit(argv[optind - 1][0])) break;
    *rad = atoi(argv[optind - 1]);
    break;
}
case 't':{
    if (!isdigit(argv[optind - 1][0])) break;
    *thickness = atoi(argv[optind - 1]);
    break;
}
case 'f':{
    *fill_flag = 1;
    break;
}
case 'p':{
    if (!isdigit(argv[optind - 1][0])) break;
    *pattern = atoi(argv[optind - 1]);
    if (*pattern < 1 || *pattern > 3) *pattern = -1;
    break;
}
case 'c':{
    int ind = optind - 1;
    int arg_len = strlen(argv[ind]);

    if (!isdigit(argv[ind][0])) break;
    int r = atoi(argv[ind]);
```

```
    if (r >= 0 && r <= 255) color1->r = r;

    int i = 0;
    for (; argv[ind][i] != '.'; i++)
        if (i >= arg_len) break;
    if (i == arg_len) break;
    i ++;

    if (!isdigit(argv[ind][i])) break;
    int g = atoi(&argv[ind][i]);
    if (g >= 0 && g <= 255) color1->g = g;

    for (; argv[ind][i] != '.'; i++)
        if (i >= arg_len) break;
    if (i == arg_len) break;
    i ++;

    if (!isdigit(argv[ind][i])) break;
    int b = atoi(&argv[ind][i]);
    if (b >= 0 && b <= 255) color1->b = b;

    break;
}
case 'C':{
    int ind = optind - 1;
    int arg_len = strlen(argv[ind]);

    if (!isdigit(argv[ind][0])) break;
    int r = atoi(argv[ind]);
    if (r >= 0 && r <= 255) color2->r = r;

    int i = 0;
    for (; argv[ind][i] != '.'; i++)
        if (i >= arg_len) break;
    if (i == arg_len) break;
    i ++;

    if (!isdigit(argv[ind][i])) break;
    int g = atoi(&argv[ind][i]);
```

```
                if (g >= 0 && g <= 255) color2->g = g;

                for (; argv[ind][i] != '.'; i++)
                    if (i >= arg_len) break;
                if (i == arg_len) break;
                i ++;

                if (!isdigit(argv[ind][i])) break;
                int b = atoi(&argv[ind][i]);
                if (b >= 0 && b <= 255) color2->b = b;

                break;
            }
            case 'o':{
                *wfile = argv[optind - 1];
            }
            default:
                break;
        }
    }
}


int readBmp(char*file_name, BmpFile *bmpf) {

    FILE *f = fopen(file_name, "rb");
    if (!f) {
        printf("Ошибка при открытии файла.\n");
        return 0;
    }


    BitmapFileHeader *bfh = &bmpf->bfh;
    BitmapInfoHeader *bih = &bmpf->bih;

    fread(bfh, sizeof(BitmapFileHeader), 1, f);
    if (bfh->signature != 0x4d42) {
        printf("Неверный формат файла.\n");
        return 0;
```

```
    }

    fread(bih, sizeof(BitmapInfoHeader), 1, f);
    if (bih->compression != 0 || bih->colorsInColorTable != 0 || bih-
>bitsPerPixel != 24) {
        printf("Непооддерживаемая версия формата.\n");
        return 0;
    }

    fseek(f, bfh->pixelArrOffset, SEEK_SET);
    unsigned int image_height = bih->height;
    unsigned int image_width = bih->width;

    Rgb **image = calloc(image_height, sizeof(Rgb *));
    unsigned long padded_width = image_width * sizeof(Rgb) + image_width
* sizeof(Rgb) % 4;
    for (int i = 0; i < image_height; i++) {
        image[i] = calloc(padded_width, 1);
        fread(image[i], 1, padded_width, f);
    }

    bmpf->pixel_array = image;

    fclose(f);
    return 1;
}

void drawLine(int x0, int y0, int x1, int y1, Rgb color, BmpFile *bf) {
    unsigned int image_height = bf->bih.height;
    unsigned int image_width = bf->bih.width;

    int dx = abs(x1 - x0), dy = abs(y1 - y0);
    int cur_x = x0, cur_y = y0;

    if (dx >= dy) {
        int err = 0;
        int derr = dy + 1;
        int x_dir = (x1 == x0) ? 0 : (x1 > x0 ? 1 : -1), y_dir = (y1 ==
y0) ? 0 : (y1 > y0 ? 1 : -1);
```

```
        for (int i = 0; i <= dx; i++) {
            cur_x += (i == 0 ? 0 : x_dir);
            if (cur_y < image_height && cur_y >= 0 && cur_x < image_width
&& cur_x >= 0)
                bf->pixel_array[cur_y][cur_x] = color;
            err = err + derr;
            if (err >= dx + 1) {
                cur_y += y_dir;
                err = err - (dx + 1);
            }
        }
    } else {
        int err = 0;
        int derr = dx + 1;
        int x_dir = (x1 == x0) ? 0 : (x1 > x0 ? 1 : -1), y_dir = (y1 ==
y0) ? 0 : (y1 > y0 ? 1 : -1);

        for (int i = 0; i <= dy; i++) {
            cur_y += (i == 0 ? 0 : y_dir);
            if (cur_y < image_height && cur_y >= 0 && cur_x < image_width
&& cur_x >= 0)
                bf->pixel_array[cur_y][cur_x] = color;
            err = err + derr;
            if (err >= dy + 1) {
                cur_x += x_dir;
                err = err - (dy + 1);
            }
        }
    }
}

void drawHexagon(int x_c, int y_c, int r, Rgb color, int thickness, int
fill, Rgb fill_color, BmpFile *bf) {
    int x[6], y[6];
    for (int i = 0; i < 6; i++) {
        double angle = M_PI / 180 * (60 * i - 30);
        x[i] = x_c + r * cos(angle);
        y[i] = y_c + r * sin(angle);
```

```
    }

    for (int j = 0; j < thickness; j++) {
        drawLine(x[0] - j, y[0], x[1] - j, y[1], color, bf);
        for (int i = 1; i < 3; i++) {
            drawLine(x[i], y[i] - j, x[i + 1], y[i + 1] - j, color, bf);
        }
        drawLine(x[3] + j, y[3], x[3] + j, y[4], color, bf);
        for (int i = 4; i < 6; i++) {
            drawLine(x[i], y[i] + j, x[(i + 1) % 6], y[(i + 1) % 6] + j,
color, bf);
        }
    }

    if (fill) {
        drawLine(x[0] - thickness, y[0], x[1] - thickness, y[1], color,
bf);
        for (int i = 1; i < 3; i++) {
            drawLine(x[i], y[i] - thickness, x[i + 1], y[i + 1] -
thickness, color, bf);
        }
        drawLine(x[3] + thickness, y[3], x[3] + thickness, y[4], color,
bf);
        for (int i = 4; i < 6; i++) {
            drawLine(x[i], y[i] + thickness, x[(i + 1) % 6], y[(i + 1) %
6] + thickness, color, bf);
        }
        drawHexagon(x_c, y_c, r - thickness, fill_color, r - thickness,
0, color, bf);
    }
}

void copyArea(int x0, int y0, int x1, int y1, int x_dest, int y_dest,
BmpFile *bf) {
    Rgb **image = bf->pixel_array;
    unsigned int image_height = bf->bih.height;
    unsigned int image_width = bf->bih.width;

    if (x0 > x1) {
```

```
        int tmp = x0;
        x0 = x1;
        x1 = tmp;
    }

    if (y0 > y1) {
        int tmp = y0;
        y0 = y1;
        y1 = tmp;
    }

    int copy_height = y1 - y0;
    int copy_width = x1 - x0;

    //copy of chosen area
    Rgb **area_copy = calloc(copy_height, sizeof(Rgb *));
    for (int i = 0; i < copy_height; i++) {
        area_copy[i] = calloc(copy_width, sizeof(Rgb));
        memcpy(area_copy[i], &image[y0 + i][x0], copy_width *
sizeof(Rgb));
    }

    //writing to destination
    for (int i = 0; i < copy_height; i++) {
        if (y_dest - i >= 0) {
            if (x_dest + copy_width < image_width)
                memcpy(&image[y_dest - i][x_dest], area_copy[copy_height
- 1 - i], copy_width * sizeof(Rgb));
            else
                memcpy(&image[y_dest - i][x_dest], area_copy[copy_height
- 1 - i],
                       (image_width - x_dest) * sizeof(Rgb));
        }
    }
    for (int i = 0; i < copy_height; i++)
        free(area_copy[i]);
    free(area_copy);
}
```

```
void changeColor(Rgb ref_color, Rgb color_to_change, BmpFile *bf) {
    unsigned int image_height = bf->bih.height;
    unsigned int image_width = bf->bih.width;
    unsigned long padded_width = image_width * sizeof(Rgb) + image_width
* sizeof(Rgb) % 4;

    for (int i = 0; i < image_height; i++) {
        for (int j = 0; j < padded_width; j++) {
            if (rgbCmp(&color_to_change, &bf->pixel_array[i][j]))
                bf->pixel_array[i][j] = ref_color;
        }
    }
}

int rgbCmp(Rgb *c1, Rgb *c2) {
    if (abs(c1->r - c2->r) < 10 && abs(c1->b - c2->b) < 10 && abs(c1->g -
c2->g) < 10)
        return 1;
    else return 0;
}

void writeBmp(char *file_name, BmpFile *bmpf) {
    unsigned int image_height = bmpf->bih.height;
    unsigned int image_width = bmpf->bih.width;
    unsigned long padded_width = image_width * sizeof(Rgb) + image_width
* sizeof(Rgb) % 4;

    FILE *wf = fopen(file_name, "wb");
    if (!wf) {printf("Ошибка при открытия файла на запись.\n"); return;}
    fwrite(&bmpf->bfh, 1, sizeof(BitmapFileHeader), wf);
    fwrite(&bmpf->bih, 1, sizeof(BitmapInfoHeader), wf);

    fseek(wf, bmpf->bfh.pixelArrOffset, SEEK_SET);

    padded_width = image_width * sizeof(Rgb) + image_width * sizeof(Rgb)
% 4;
    for (int i = 0; i < image_height; i++) {
        fwrite(bmpf->pixel_array[i], 1, padded_width, wf);
    }
```

```
    for (int i = 0; i < image_height; i++)
        free(bmpf->pixel_array[i]);
    free(bmpf->pixel_array);
    fclose(wf);
}


void gradientFrame(Rgb start_color, Rgb end_color, int frame_width,
BmpFile *bf) {
    int image_width = (int) bf->bih.width - 1;
    int image_height = (int) bf->bih.height - 1;

    int r_step = (end_color.r - start_color.r) / frame_width;
    int r_rem = (end_color.r - start_color.r) % frame_width;

    int g_step = (end_color.g - start_color.g) / frame_width;
    int g_rem = (end_color.g - start_color.g) % frame_width;

    int b_step = (end_color.b - start_color.b) / frame_width;
    int b_rem = (end_color.b - start_color.b) % frame_width;

    for (int i = 0; i < frame_width; i++) {
        start_color.r += r_step + (i > frame_width - r_rem ? 1 : 0);
        start_color.g += g_step + (i > frame_width - g_rem ? 1 : 0);
        start_color.b += b_step + (i > frame_width - b_rem ? 1 : 0);
        drawLine(0 + i, 0 + i, 0 + i, image_height - i, start_color, bf);
        drawLine(0 + i, image_height - i, image_width - i, image_height -
i, start_color, bf);
        drawLine(image_width - i, image_height - i, image_width - i, 0 +
i, start_color, bf);
        drawLine(image_width - i, 0 + i, 0 + i, 0 + i, start_color, bf);
    }
}

void stripedFrame(Rgb color1, Rgb color2, int frame_width, BmpFile *bf) {
    int image_width = (int) bf->bih.width - 1;
    int image_height = (int) bf->bih.height - 1;
```

```
    Rgb colors[] = {color1, color2};
    for (int i = 0; i < frame_width; i++) {
        drawLine(0 + i, 0 + i, 0 + i, image_height - i, colors[i % 2],
bf);
        drawLine(0 + i, image_height - i, image_width - i, image_height -
i, colors[i % 2], bf);
        drawLine(image_width - i, image_height - i, image_width - i, 0 +
i, colors[i % 2], bf);
        drawLine(image_width - i, 0 + i, 0 + i, 0 + i, colors[i % 2],
bf);
    }
}

void wavyFrame(Rgb color, int thickness, BmpFile *bf) {
    int image_width = (int) bf->bih.width;
    int image_height = (int) bf->bih.height;

    for (int x = 0; x < image_width; x++) {
        for (int y = 0; y < thickness; y++) {
            int f_x = (int) fabs(thickness * sin(2.0 / thickness * x));
            if (y <= f_x)
                bf->pixel_array[y][x] = color;
        }
    }

    for (int y = 0; y < image_height; y++) {
        for (int x = (int) image_width - thickness; x < image_width; x++)
{
            int f_y = (int) fabs(thickness * sin(2.0 / thickness * y));
            if (image_width - x <= f_y)
                bf->pixel_array[y][x] = color;
        }
    }

    for (int x = 0; x < image_width; x++) {
        for (int y = image_height - thickness; y < image_height; y++) {
            int f_x = image_height - (int) fabs(thickness * sin(2.0 /
thickness * x));
            if (y >= f_x)
```

```
                bf->pixel_array[y][x] = color;
            }
        }

        for (int y = 0; y < image_height; y++) {
            for (int x = 0; x < thickness; x++) {
                int f_y = (int) fabs(thickness * sin(2.0 / thickness * y));
                if (x <= f_y)
                    bf->pixel_array[y][x] = color;
            }
        }
    }
```