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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Обработка BMP файла

Студент гр. 3341 _____ Костромитин М.М.

Преподаватель _____ Глазунов С.А.

Санкт-Петербург

2024

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Костромитин М.М.

Группа 3341

Вариант 2

Программа обязательно должна иметь CLI (опционально дополнительное использование GUI). Более подробно тут:
http://se.moevm.info/doku.php/courses:programming:rules_extra_kurs

Программа должна реализовывать весь следующий функционал по обработке bmp-файла

Общие сведения

24 бита на цвет

без сжатия

файл может не соответствовать формату BMP, т.е. необходимо проверка на BMP формат (дополнительно стоит помнить, что версий у формата несколько). Если файл не соответствует формату BMP или его версии, то программа должна завершиться с соответствующей ошибкой.

обратите внимание на выравнивание; мусорные данные, если их необходимо дописать в файл для выравнивания, должны быть нулями.

обратите внимание на порядок записи пикселей

все поля стандартных BMP заголовков в выходном файле должны иметь те же значения что и во входном (разумеется кроме тех, которые должны быть изменены).

Программа должна иметь следующую функции по обработке изображений:

(1) Заменяет все пиксели одного заданного цвета на другой цвет. Флаг для выполнения данной операции: `--color_replace`. Функционал определяется:

Цвет, который требуется заменить. Флаг `--old_color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--old_color 255.0.0` задаёт красный цвет)
Цвет на который требуется заменить. Флаг `--new_color` (работает аналогично флагу `--old_color`)
(2) Фильтр rgb-компонент. Флаг для выполнения данной операции: `--rgbfilter`. Этот инструмент должен позволять для всего изображения либо установить в диапазоне от 0 до 255 значение заданной компоненты.
Функционал определяется
Какую компоненту требуется изменить. Флаг `--component_name`.
Возможные значения `red`, `green` и `blue`.
В какой значение ее требуется изменить. Флаг `--component_value`.
Принимает значение в виде числа от 0 до 255
(3) Разделяет изображение на N*M частей. Флаг для выполнения данной операции: `--split`. Реализация: провести линии заданной толщины.
Функционал определяется:
Количество частей по “оси” Y. Флаг `--number_x`. На вход принимает число больше 1
Количество частей по “оси” X. Флаг `--number_y`. На вход принимает число больше 1
Толщина линии. Флаг `--thickness`. На вход принимает число больше 0
Цвет линии. Флаг `--color` (цвет задаётся строкой `rrr.ggg.bbb`, где rrr/ggg/bbb – числа, задающие цветовую компоненту. пример `--color 255.0.0` задаёт красный цвет)
Все подзадачи, ввод/вывод должны быть реализованы в виде отдельной функции.

Содержание пояснительной записки:
разделы «Аннотация», «Содержание», «Введение», «Ход работы», «Пример работы программы», «Заключение», «Список использованных источников»

Предполагаемый объем пояснительной записки:
Не менее 15 страниц.

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 27.05.2024

Дата защиты реферата: 29.05.2024

Студент		Костромитин М.М.
Преподаватель		Глазунов С.А.

АННОТАЦИЯ

В данной курсовой работе была реализована программа, обрабатывающая PNG изображения, не имеющие сжатия. Программа проверяет тип изображения, его версию, при соответствии требованиям в дальнейшем обрабатывает его и подаёт на выход изменённую копию изображения. Взаимодействие с программой осуществляется с помощью CLI (интерфейс командной строки).

SUMMARY

In this course has been created a program that processes uncompressed PNG images. The program checks the type of image, its version, if it meets the requirements, it further processes it and outputs a modified copy of the image. Interaction with the program is performed using CLI (command line interface).

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью курсовой работы является изучение форматов файлов BMP и PNG, а также реализация функций для работы с этими форматами файлов.

Для достижения поставленной цели требуется решить следующие задачи:

1) изучить BMP и PNG форматы изображений;

2) получить информацию об изображении: размеры, содержимое и др.;

3) обработать массив пикселей в соответствии с заданием;

4) обработать исключительные случаи;

5) сохранить итоговое изображение в новый файл.

1. ХОД ВЫПОЛНЕНИЯ РАБОТЫ

1.1 Ввод начальных данных

Обработка входных данных осуществляется в функции int main(int argc, char* argv[]), в которой создается массив struct option longopts[] хранящий информацию о длинных флагах. Далее с помощью функции getopt_long идет последовательная обработка входных данных: проверяется наличие и корректность введенных аргументов для флагов, а также после обработки всех флагов проверяется наличие зависимых флагов для необходимых функций, и если все необходимые флаги присутствуют, то происходит выполенние нужной функции (например для выполнения функции void rgb_filter проверяется наличие флагов –component_name и –component_value). После обработки или во время обработки флагов (в случае наличия флага --input) в функции Rgb** read_bmp происходит проверка того, что файл является файлом формата BMP, а после идет считывание файла: считывается заголовок файла (в структуру BitmapFileHeader), считывается заголовок с информацией (в структуру BitmapInfoHeader), считывается и возвращается функцией массив пикселей (в Rgb** arr).

1.2 Функции курсовой работы

Rgb** read_bmp (char file_name[], BitmapFileHeader* bmfh, BitmapInfoHeader* bmif) – функция считывания BMP файла, возвращает массив пикселей переданной картинки.

int check_if_bmp (BitmapFileHeader* bmfh) – по заголовку файла проверяет - является ли входной файл файлом формата BMP.

void write_bmp(char file_name[], Rgb **arr, int H, int W, BitmapFileHeader bmfh, BitmapInfoHeader bmif) – записывает результат в файл с именем file_name.

void color_replace(Rgb** arr, int H, int W, Rgb old_color, Rgb new_color) – заменяет цвет old_color на цвет new_color в изображении.

int color_cmp(Rgb f, Rgb s) – возвращает 1, если структуры Rgb f и s совпадают, иначе возвращает 0.

void color_change(Rgb* dest, Rgb src) – записывает в структуру Rgb dest значения структуры Rgb src, то есть изменяет цвет dest на цвет src.

void rgb_filter(Rgb** arr, int H, int W, char component[], int value) – изменяет значение некоторой компоненты (red, green или blue) на значение value для всех пикселей картинки.

void draw_line(Rgb** arr, int H, int W, int x1, int x2, int y1, int y2, Rgb color) – рисует произвольную линию по алгоритму Брезенхема толщины 1.

void split(Rgb** arr, int horizontal, int vertical, int H, int W, int thickness, Rgb color) – чертит горизонтальные линии толщиной thickness и цвета color в количестве horizontal, а также вертикальные линии толщиной thickness и цвета color в количестве vertical, так, чтобы картинки поделилась на равные части по вертикали и горизонтали.

int color_validator(Rgb color) – возвращает 0, если значения всех компонент цвета удовлетворяют формату RGB (то есть больше или рано 0 и меньше либо равно 255), иначе возвращает 1.

int coord_validator(int max, int coord0, int coord1) – возвращает 0, если координаты по x или по y произвольной линии находятся в пределах границ изображения, иначе возвращает 1.

void swap_int(int* a, int* b) – помещает в переменную a значение переменной b, а в переменную b значение переменной a.

unsigned int padding(unsigned int w) – возвращает значения сдвига для изображения.

unsigned int row_len(unsigned int w) – возвращает количество пикселей в горизонтальном ряду с учетом сдвига.

void help() – выводит на экран справку о программе.

void info(BitmapInfoHeader info_header, BitmapFileHeader file_header) – выводит на экран информацию о файле BMP.

int component_validator(int component) – возвращает 1, если значения одной из компоненты RGB меньше либо равна 255 и больше либо равна 0, иначе возвращает 0.

Rgb parse_color(char color_string[]) – преобразует строчку вида “rrr.ggg.bbb” в структуру Rgb и возвращает ее.

void print_file_header(BitmapFileHeader header) – выводит на экран информацию из заголовка файла.

void print_info_header(BitmapInfoHeader header) – выводит на экран информацию из заголовка с информацией.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были изучены формата файлов изображений BMP, а также разработаны функции для работы с ними. Подробное изучение особенностей и структуры данных данных форматов позволило успешно реализовать функционал, включающий получение информации об изображениях, обработку массива пикселей в соответствии с поставленными задачами, учет и обработку исключительных случаев, а также сохранение результирующего изображения в новом файле. А также была разобрана задача обработки флагов CLI с помощью функции getopt_long из библиотеки getopt.h.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1.https://se.moevm.info/lib/exe/fetch.php/courses:programming:programming_cw_metoda_2nd_course_last_ver.pdf.pdf - методические материалы для написания курсовой работы

2.https://www.r5.org/files/books/computers/languages/c/kr/Brian_Kernighan_Dennis_Ritchie-The_C_Programming_Language-RU.pdf – язык программирования Си

3.https://www.gnu.org/software/libc/manual/html_node/Getopt.html - принцип работы функции getopt_long и примеры обработки флагов с помощью неё.

4. https://ru.wikibooks.org/wiki/Реализации_алгоритмов/Алгоритм_Брезенхэма - алгоритм Брезенхэма

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <getopt.h>
#include <unistd.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)


Rgb **read_bmp(char file_name[], BitmapFileHeader* bmfh, BitmapInfoHeader*
bmif);
int check_if_bmp(BitmapFileHeader* bmfh);
void write_bmp(char file_name[], Rgb **arr, int H, int W, BitmapFileHeader bmfh,
BitmapInfoHeader bmif);
void color_replace(Rgb** arr, int H, int W, Rgb old_color, Rgb new_color);
int color_cmp(Rgb f, Rgb s);
void color_change(Rgb* dest, Rgb src);
void rgb_filter(Rgb** arr, int H, int W, char component[], int value);
void draw_line(Rgb** arr, int H, int W, int x0, int x1, int y0, int y1, Rgb
color);
void split(Rgb** arr, int horizontal, int vertical, int H, int W, int thickness,
Rgb color);
int color_validator(Rgb color);
int coord_validator(int max, int coord0, int coord1);
void swap_int(int* a, int* b);
unsigned int padding(unsigned int w);
```

```
unsigned int row_len(unsigned int w);
Rgb parse_color(char color_string[]);
void help();
void info(BitmapInfoHeader info_header, BitmapFileHeader file_header);
void print_file_header(BitmapFileHeader header);
void print_info_header(BitmapInfoHeader header);


int
main(int argc, char* argv[]){
    printf("Course work for option 4.2, created by Kostromitin Mikhail\n");

    BitmapFileHeader file_header;
    BitmapInfoHeader file_info;

    int is_replace = -1;
    int is_filter = -1;
    int is_split = -1;

    struct option longopts[] = {
            {"help", no_argument, NULL, 'h'},
            {"output", required_argument, NULL, 'o'},
            {"input", required_argument, NULL, 'p'},
            {"info", no_argument, NULL, 'i'},
            {"color_replace", no_argument, &is_replace, 'r'},
            {"old_color", required_argument, NULL, 'a'},
            {"new_color", required_argument, NULL, 'w'},
            {"rgbfilter", no_argument, &is_filter, 'f'},
            {"component_name", required_argument, NULL, 'n'},
            {"component_value", required_argument, NULL, 'v'},
            {"split", no_argument, &is_split, 's'},
            {"number_x", required_argument, NULL, 'x'},
            {"number_y", required_argument, NULL, 'y'},
            {"thickness", required_argument, NULL, 't'},
            {"color", required_argument, NULL, 'c'},
            {0, 0, 0, 0}
    };

    int gpt;
    char shortopts[] = "ho:";
    int* indexptr = (int*)malloc(sizeof(int));
    char file_name_new[256];
    file_name_new[0] = '\0';

    char file_name_old[100];
    strncpy(file_name_old, argv[argc - 1], strlen(argv[argc - 1]));

    char colors_replace_old[12];
    char colors_replace_new[12];
    Rgb replace_old;
    Rgb replace_new;

    char component_name[6];
    int component_value;

    int x;
    int y;
    int th;
    char split_color[12];
    Rgb clr;

    int is_read = 0;
```

```
    Rgb** arr;

    while ((gpt = getopt_long(argc, argv, shortopts, longopts, indexptr)) != -
1){
        switch(gpt){
            case 'h':
                help();
                exit(0);
            case  'o':
                strncpy(file_name_new, optarg, strlen(optarg));
                file_name_new[strlen(optarg)] = '\0';
                break;
            case 'p':
                strncpy(file_name_old, optarg, strlen(optarg));
                file_name_old[strlen(optarg)] = '\0';
                arr = read_bmp(file_name_old, &file_header, &file_info);
                is_read = 1;
                break;
            case 'i':
                info(file_info, file_header);
                exit(0);
            case 'a':
                strncpy(colors_replace_old, optarg, strlen(optarg));
                colors_replace_old[strlen(optarg)] = '\0';
                replace_old = parse_color(colors_replace_old);
                break;
            case 'w':
                strncpy(colors_replace_new, optarg, strlen(optarg));
                colors_replace_new[strlen(optarg)] = '\0';
                replace_new = parse_color(colors_replace_new);
                break;
            case 'n':
                strncpy(component_name, optarg, strlen(optarg));
                component_name[strlen(optarg)] = '\0';
                break;
            case 'v':
                component_value = atoi(optarg);
                break;
            case 'x':
                x = atoi(optarg);
                break;
            case 'y':
                y = atoi(optarg);
                break;
            case 't':
                th = atoi(optarg);
                break;
            case 'c':
                strncpy(split_color, optarg, strlen(optarg));
                split_color[strlen(optarg)] = '\0';
                clr = parse_color(split_color);
                break;
            default:
                break;
        }
    }

    if (!is_read) {
        arr = read_bmp(file_name_old, &file_header, &file_info);
    }
    int H, W;
    H = file_info.height;
```

```
    W = file_info.width;

    if (is_replace != -1){
        color_replace(arr, H, W, replace_old, replace_new);
    }
    if (is_filter != -1){
        rgb_filter(arr, H, W, component_name, component_value);
    }
    if (is_split != -1){
        split(arr, x, y, H, W, th, clr);
    }

    if (file_name_new[0] == '\0'){
        write_bmp(file_name_old, arr, H, W, file_header, file_info);
    } else{
        write_bmp(file_name_new, arr, H, W, file_header, file_info);
    }

    for (int i = 0; i < H; i++)
        free(arr[i]);
    free(arr);
    free(indexptr);
    return 0;
}


Rgb**
read_bmp(char file_name[], BitmapFileHeader* bmfh, BitmapInfoHeader* bmif){
    FILE *f = fopen(file_name, "rb");
    fread(bmfh, 1, sizeof(BitmapFileHeader), f);
    fread(bmif, 1, sizeof(BitmapInfoHeader), f);
    if (!check_if_bmp(bmfh)){
        printf("Error 49: invalid file format\n");
        exit(49);
    }
    unsigned int H = bmif->height;
    unsigned int W = bmif->width;
    Rgb** arr = malloc(H * sizeof(Rgb*));
    for(int i = 0; i < H; i++){
        arr[i] = malloc(row_len(W));
        fread(arr[i], 1, row_len(W), f);
    }
    fclose(f);
    return arr;
}


int
check_if_bmp(BitmapFileHeader* bmfh){
    if (bmfh->signature != 19778 || bmfh->reserved1 != 0 || bmfh->reserved2 !=
0){
        return 0;
    }
    return 1;
}


void
write_bmp(char file_name[], Rgb **arr, int H, int W, BitmapFileHeader bmfh,
BitmapInfoHeader bmif){
    FILE *ff = fopen(file_name, "wb");
    fwrite(&bmfh, 1, sizeof(BitmapFileHeader), ff);
```

```
    fwrite(&bmif, 1, sizeof(BitmapInfoHeader), ff);
    for(int i = 0; i < H; i++){
        fwrite(arr[i], 1, row_len(W), ff);
    }
    fclose(ff);
}


void
color_replace(Rgb** arr, int H, int W, Rgb old_color, Rgb new_color){
    if (!(color_validator(old_color)) || !(color_validator(new_color))){
        printf("Error 41: invalid color parameters");
        exit(41);
    }
    for (int y = 0; y < H; y++){
        for (int x = 0; x < W; x++){
            if (color_cmp(arr[y][x], old_color)){
                color_change(&arr[y][x], new_color);
            }
        }
    }
}


int
color_cmp(Rgb f, Rgb s){
    if (f.g == s.g && f.r == s.r && f.b == s.b){
        return 1;
    }
    return 0;
}


void
color_change(Rgb* dest, Rgb src){
    (*dest).b = src.b;
    (*dest).g = src.g;
    (*dest).r = src.r;
}


void
rgb_filter(Rgb** arr, int H, int W, char component[], int value){
    if (value < 0 || value > 255){
        printf("Error 42: invalid component value");
        exit(42);
    }
    if (!strcmp(component, "red")){
        for (int y = 0; y < H; y++){
            for (int x = 0; x < W; x++){
                arr[y][x].r = value;
            }
        }
    } else if (!strcmp(component, "green")){
        for (int y = 0; y < H; y++){
            for (int x = 0; x < W; x++){
                arr[y][x].g = value;
            }
        }
    } else if (!strcmp(component, "blue")){
        for (int y = 0; y < H; y++){
            for (int x = 0; x < W; x++){
```

```
                arr[y][x].b = value;
            }
        }
    } else {
        printf("Error 42: invalid component");
        exit(42);
    }
}


void
draw_line(Rgb** arr, int H, int W, int x1, int x2, int y1, int y2, Rgb color) {
    if (!coord_validator(W, x1, x2) || !coord_validator(H, y1, y2) ||
!(color_validator(color))){
        printf("Error 43: invalid parameters for a line");
        exit(43);
    }
    const int deltaX = abs(x2 - x1);
    const int deltaY = abs(y2 - y1);
    const int signX = x1 < x2 ? 1 : -1;
    const int signY = y1 < y2 ? 1 : -1;
    int error = deltaX - deltaY;
    arr[y2- 1][x2 - 1] = color;
    while(x1 != x2 || y1 != y2){
        arr[y1][x1] = color;
        int error2 = error * 2;
        if(error2 > -deltaY){
            error -= deltaY;
            x1 += signX;
        }
        if(error2 < deltaX){
            error += deltaX;
            y1 += signY;
        }
    }

}


void
split(Rgb** arr, int horizontal, int vertical, int H, int W, int thickness, Rgb
color){
    if (horizontal < 2 || vertical < 2 || thickness <= 0){
        printf("Error 44: invalid split parameters");
        exit(44);
    }

    int y = 0;
    for (int i = 1; i < horizontal; i++){
        y += H / horizontal;
        int y_for_thick = y;

        for (int thick = 0; thick < thickness; thick++){
            if (thick % 2 == 0){
                y_for_thick -= thick;
                draw_line(arr, H, W, 0, W, y_for_thick, y_for_thick, color);
            } else {
                y_for_thick += thick;
                draw_line(arr, H, W, 0, W, y_for_thick, y_for_thick, color);
            }
        }
    }
```

```
    int x = 0;
    for (int i = 1; i < vertical; i++){
        x += W / vertical;
        int x_for_thick = x;

        for (int thick = 0; thick < thickness; thick++){
            if (thick % 2 == 0){
                x_for_thick -= thick;
                draw_line(arr, H, W, x_for_thick, x_for_thick, 0, H, color);
            } else {
                x_for_thick += thick;
                draw_line(arr, H, W, x_for_thick, x_for_thick, 0, H, color);
            }
        }
    }
}


int
color_validator(Rgb color){
    if (color.r < 0 || color.r > 255 || color.g < 0 || color.g > 255 || color.b
< 0 || color.b > 255){
        return 0;
    }
    return 1;
}


int
coord_validator(int max, int coord0, int coord1){
    if (coord0 < 0 || coord0 > max || coord1 < 0 || coord1 > max){
        return 0;
    }
    return 1;
}


void
swap_int(int* a, int* b){
    int tmp = *a;
    *a = *b;
    *b = tmp;
}


unsigned int
padding(unsigned int w){
    unsigned int padding = (w * sizeof(Rgb)) % 4;
    if (padding) padding = 4 - padding;
    return padding;
}


unsigned int
row_len(unsigned int w){
    return w * sizeof(Rgb) + padding(w);
}


void
help(){
```

```
    printf("Программа для обработки файла формата bmp.\n");
    printf("\t-p / --input - имя входного файла.\n");
    printf("\t-i / --info - флаг для печати подробной информации о bmp-
файле.\n");
    printf("\t-o / --output - переопределение названия конечного bmp-файла.\n");
    printf("Заменяет все пиксели одного заданного цвета на другой цвет. Флаг для
выполнения данной операции: `--color_replace`. Функционал определяется:\n");
    printf("\t--old_color - цвет, который необходимо заменить: строчка вида
\"rrr.ggg.bbb\", где rrr/ggg/bbb – числа, задающие цветовую компоненту.\n");
    printf("\t--new_color - новый цвет: строчка вида \"rrr.ggg.bbb\", где
rrr/ggg/bbb – числа, задающие цветовую компоненту.\n");
    printf("Фильтр rgb-компонент. Флаг для выполнения данной операции: `--
rgbfilter`. Этот инструмент должен позволять для всего изображения либо
установить в диапазоне от 0 до 255 значение заданной компоненты. Функционал
определяется:\n");
    printf("\t--component_name - имя компоненты, которую нужно заменить.
Возможные значения `red`, `green` и `blue`\n");
    printf("\t--component_value - значение, на которое требуется заменить
значения компоненты. Принимает значение в виде числа от 0 до 255.\n");
    printf("Разделяет изображение на N*M частей. Флаг для выполнения данной
операции: `--split`. Реализация: провести линии заданной толщины. Функционал
определяется:\n");
    printf("\t--number_x - количество частей по “оси” Y. На вход принимает число
больше 1.\n");
    printf("\t--number_y - количество частей по “оси” X. На вход принимает число
больше 1.\n");
    printf("\t--thickness - толщина линии. На вход принимает число больше
0.\n");
    printf("\t--color - цвет линии, цвет задаётся строкой \"rrr.ggg.bbb\", где
rrr/ggg/bbb – числа, задающие цветовую компоненту.\n");
    printf("При последовательном вводе операция, они имеют следующий приоритет
выполнения: color_replace, rgbfilter, split.\n");
}


void
info(BitmapInfoHeader info_header, BitmapFileHeader file_header){
    print_file_header(file_header);
    print_info_header(info_header);
}


int
component_validator(int component){
    if (component > 255 || component < 0){
        return 1;
    }
    return 0;
}


Rgb
parse_color(char color_string[]){
    Rgb color;
    char* tmp;
    int curr_color;

    tmp = strtok(color_string, ".");
    curr_color = atoi(tmp);
    if (component_validator(curr_color)){
        printf("Error 45: Invalid component for color replacement");
        exit(45);
```

```
    }
    color.r = curr_color;

    tmp = strtok(NULL, ".");
    curr_color = atoi(tmp);
    if (component_validator(curr_color)){
        printf("Error 45: Invalid component for color replacement");
        exit(45);
    }
    color.g = curr_color;

    tmp = strtok(NULL, ".");
    curr_color = atoi(tmp);
    if (component_validator(curr_color)){
        printf("Error 45: Invalid component for color replacement");
        exit(45);
    }
    color.b = curr_color;

    return color;
}


void
print_file_header(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
print_info_header(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);
}
```

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ

Фото для обработки – RGB фильтр

```
./cw --input My-boy-be-ballin.bmp --rgbfilter --component_name blue --
component_value 0
```

Результат работы программы:

Фото для обработки – замена одного цвета на другой

```
./cw --input Plant.bmp --color_replace --old_color 000.000.000 --new_color
255.255.255
```

Результат работы программы:

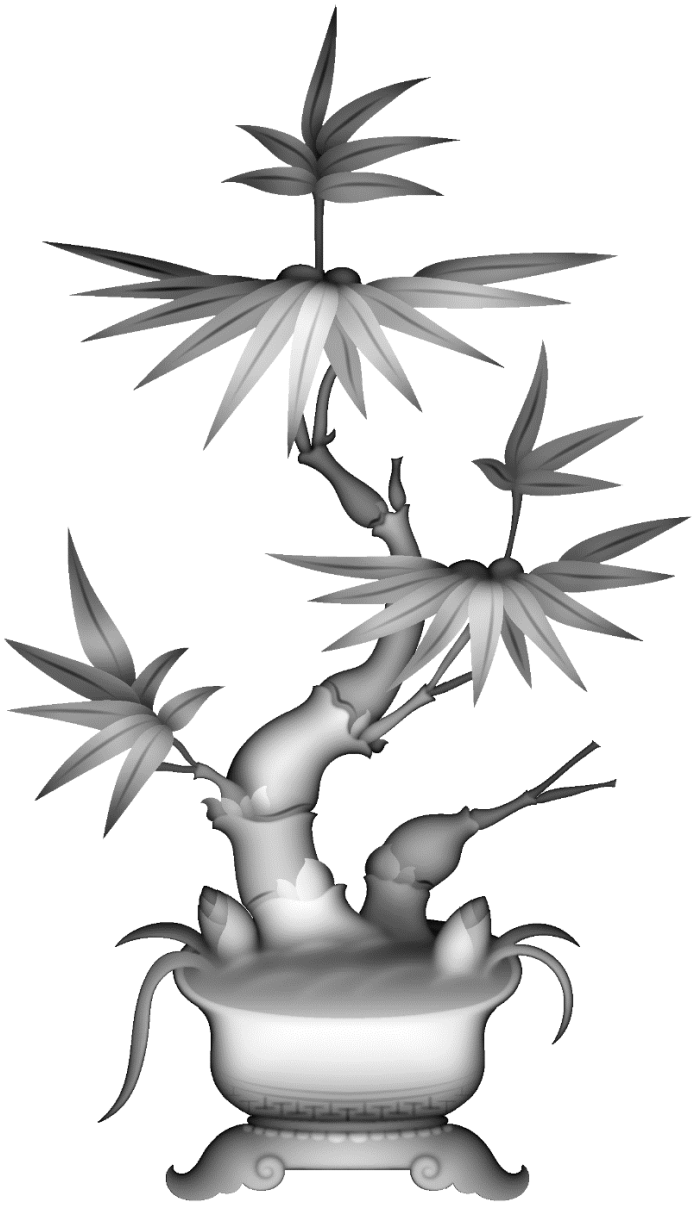

Фото для обработки – разделение картинки на равные части

```
./cw --input Obama.bmp --split --number_x 10 --number_y 10 --thickness 20
--color 0.0.0
```

Результат работы программы: