

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

КУРСОВАЯ РАБОТА
по дисциплине «Программирование»
Тема: Работа с изображениями в Си

Студент гр. 3385

Комаренко Т. А.

Преподаватель

Глазунов С. А.

Санкт-Петербург

2024

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент: Комаренко Т. А.

Группа: 3385

Тема работы: Работа с изображениями в Си.

Исходные данные:

Вариант 3.5

Программа обязательно должна иметь **CLI** (опционально дополнительное использование GUI). Программа должна реализовывать весь следующий функционал по обработке bmp-файла

Общие сведения

- 24 бита на цвет
- без сжатия
- файл всегда соответствует формату BMP (но стоит помнить, что версий у формата несколько)
- обратите внимание на выравнивание; мусорные данные, если их необходимо дописать в файл для выравнивания, должны быть нулями.
- обратите внимание на порядок записи пикселей
- все поля стандартных BMP заголовков в выходном файле должны иметь те же значения что и во входном (разумеется кроме тех, которые должны быть изменены).

Программа должна иметь следующую функции по обработке изображений:

- Установить все компоненты пикселя как среднее арифметическое из них (округление вниз). Флаг для выполнения данной операции: `--

`component_mean``. Т.е. если пиксель имеет цвет (70, 140, 220), то после применения операции цвет будет (143, 143, 143)

- Установить все компоненты пикселя как минимальную из них. Флаг для выполнения данной операции: `--component_min``. Т.е. если пиксель имеет цвет (150, 100, 200), то после применения операции цвет будет (100, 100, 100)

Содержание пояснительной записки:

«Аннотация», «Содержание», «Введение», «Изображение», «Ввод опций», «Функции для обработки изображений», «Makefile», «Заключение», «Список использованных источников», «Приложение А. Пример работы программы», «Приложение В. Исходный код программа»

Предполагаемый объем пояснительной записки:

Не менее 20 страниц.

Дата выдачи задания: 18.03.2024

Дата сдачи реферата: 23.05.2024

Дата защиты реферата: 25.05.2024

Студент _____ Комаренко Т. А.

Преподаватель _____ Глазунов С. А.

АННОТАЦИЯ

В ходе курсовой работы требуется разработать программу на языке Си для обработки изображений в формате BMP. Для хранения изображения предлагается создать структуру, соответствующую спецификации формата BMP.

Программа должна считывать аргументы, введенные пользователем. Если аргументы введены некорректно, сообщить об ошибке, иначе обработать изображение требуемым образом.

Пример работы программы приведен в приложении А.

Исходный код программы приведен в приложении Б.

SUMMARY

During the course work, it is required to develop a C program for image processing in BMP format. To store an image, it is proposed to create a structure corresponding to the BMP format specification.

The program must read the arguments entered by the user. If the arguments are entered incorrectly, report an error, otherwise process the image in the required way.

An example of how the program works is given in Appendix A.

The source code of the program is given in Appendix B.

СОДЕРЖАНИЕ

	Введение	6
1.	Изображение	7
1.1	Структуры BitmapFileHeader и BitmapInfoHeader	7
1.2	Загрузка изображения	8
1.3	Сохранение изображения	8
2.	Ввод опций	9
2.1	Структура option	9
2.2	Использование getopt_long	9
3.	Функции для обработки изображений	11
3.1	Функция comp_mean	11
3.2	Функция comp_min	11
3.3	Организация вызова этих функций	11
4	Makefile	12
	Заключение	13
	Список использованных источников	14
	Приложение А. Пример работы программы	15
	Приложение В. Исходный код программы	16

ВВЕДЕНИЕ

Цель работы: написать программу на языке Си, которая считывает изображение и обрабатывает его требуемым пользователем образом. Для этого требуется реализовать:

- Загрузку изображения из файла;
- Сохранение изображения в файл;
- Считывание аргументов из командной строки;
- Обработку некорректного ввода;
- Попиксельную обработку изображения;

1. ИЗОБРАЖЕНИЕ

1.1 Структуры **BitmapFileHeader** и **BitmapInfoHeader**.

Изображение формата BMP состоит из двух заголовков. Самый первый (**BitmapFileHeader**) состоит из:

1. Сигнатуры файла. 2 байта. Для BMP файла они равны BM (signature).
2. Размер файла в байтах. 4 байта (filesize).
3. Зарезервированных 4 байтов (reserved1/reserved2).
4. Смещения от начала файла в байтах, в котором находятся данные о цветах пикселей изображения (pixelArrOffset).

Далее идет заголовок с другой информацией об изображении

(**BitmapInfoHeader**):

1. Размер заголовка. 4 байта (headersize).
2. Ширина изображения. 4 байта (width).
3. Высота изображения. 4 байта (height).
4. Число плоскостей. 2 байта (planes).
5. Число бит на пиксель. 2 байта (bitsPerPixel).
6. Вид сжатия. 4 байта. Равен нулю, если изображение сохранено без сжатия (compression).
7. Количество пикселей на метр по оси X. 4 байта (xPixelsPerMeter).
8. Количество пикселей на метр по оси Y. 4 байта (yPixelsPerMeter).
9. Число цветов в цветовой таблице (при использовании RGB палитра отсутствует, поэто число цветов равно нулю). 4 байта (colorsInColorTable).
10. Число важных цветов в цветовой таблице. 4 байта (importantColorCount).

Эти заголовки описаны структурами **BitmapFileHeader** и **BitmapInfoHeader** соответственно.

Чтобы отключить выравнивание структур, которое по умолчанию включено в Си, можно использовать атрибут `packed` для структур.

Структура **Rgb** состоит из трех байтов. По одному на компоненту цвета.

1.2 Загрузка изображения

Функция `read_bmp` предназначена для загрузки изображения из BMP-файла. Сначала она открывает файл в бинарном режиме чтения («rb»). Если открыть файл не удастся, функция выводит сообщение об ошибке и возвращает соответствующий код ошибки.

Далее функция считывает заголовки из файла. Если чтение заголовков не удастся, она выводит сообщение об ошибке, закрывает файл и возвращает код ошибки. Затем функция проверяет сигнатуру файла, чтобы убедиться, что это действительно BMP-файл.

После успешной проверки сигнатуры функция выделяет память для массива пикселей и считывает пиксели из файла.

В конце файл закрывается, функция возвращает указатель на структуру типа `Rgb`.

1.3 Сохранение изображения

За сохранение изображения отвечает функция `write_bmp`. Сначала открывается нужный файл для двоичной записи (режим «wb»).

Далее записываются оба заголовка.

Далее файл закрывается.

Чтобы освободить память, используемую изображением, создана функция `freeMemory`, которая высвобождает память, используемую для массива пикселей и заголовочных файлов.

2. ВВОД ОПЦИЙ

2.1 Структура option

Эта структура используется для описания опций командной строки в программе на языке C. Структура option определена в заголовочном файле `<getopt.h>` и используется функцией `getopt_long` для обработки длинных опций.

Каждый элемент массива `long_options` описывает одну опцию командной строки и имеет следующую структуру:

- `name`: строка, содержащая имя опции (например, "input" или "output").
- `has_arg`: указывает, требует ли опция аргумент. Возможные значения:
- `no_argument (0)` — опция не требует аргумент.
- `required_argument (1)` — опция требует аргумент.
- `optional_argument (2)` — опция может иметь необязательный аргумент.
- `flag`: указатель на переменную типа `int`, который используется для хранения флага, указывающего, была ли опция передана. Если `flag` равен `NULL`, то код опции будет возвращен функцией `getopt_long`.
- `val`: значение, которое будет возвращено функцией `getopt_long` при обнаружении этой опции. Если `flag` не равен `NULL`, то это значение будет присвоено переменной, на которую указывает `flag`.

2.2 Использование getopt_long

Функция `getopt_long` из стандартной библиотеки `getopt.h` в языке программирования Си предназначена для обработки длинных и коротких опций командной строки. Она позволяет разбирать аргументы командной строки по определенным флагам и их значениям. Функция `getopt_long` принимает следующие аргументы:

- `argc`: количество аргументов командной строки.
- `argv`: массив строк, содержащий сами аргументы командной строки.
- `shortopts`: строка, определяющая короткие опции (флаги).
- `longopts`: массив структур `option`, определяющий длинные опции (флаги).

Функция `getopt_long` возвращает следующие значения:

- `-1`: если все опции были считаны.
- `«?»`: если произошла ошибка при считывании опции.
- `val`: значение опции, если опция успешно считана. В случае короткой опции `val` равно символу считанного флага, в случае длинной опции `val` равно определенному в `longopts` числу.

3. ФУНКЦИИ ДЛЯ ОБРАБОТКИ ИЗОБРАЖЕНИЙ

3.1 Функция `comp_mean`

Эта функция отвечает за замену цвета. В качестве аргументов принимает указатель на изображение, длину и ширину изображения взятые из заголовочных файлов. Она проходится по каждому пикселю изображения и определяет среднее значение для трех компонент каждого пикселя. Далее присваивает эти средние значения соответствующим пикселям.

3.2 Функция `comp_min`

Эта функция схожа с функцией `comp_mean`, она проходится по всем пикселям изображения и для каждого пикселя определяет минимальную компоненту, после чего присваивает каждой компоненте этого пикселя данное значение.

3.3 Организация вызова этих функций

В функции `main` после считывания и записи изображения и информации о нем организована структура `option` и с помощью функции `getopt_long_only` происходит считывания опций.

После чего происходит проверка на пустой список опций которые были поданы пользователем. Далее с помощью цикла `while` пока команда `getopt_long_only` не пройдет по всем опциям происходит проверка на нахождение каждой возможной опции среди тех которые ввел пользователь, при соответствии значение соответствующего флага меняется, после чего происходит проверка каждого флага и вызов функций.

MAKEFILE

Для сборки программы был использован Makefile. Необходимость использовать MAKEFILE, заключается в использовании функции powf, а точнее использование флага компиляции -lm.

ЗАКЛЮЧЕНИЕ

В ходе выполнения была реализована программа, которая считывает изображение введенное пользователем и редактирует его в соответствии с действиями пользователя.

- Загрузка изображения из файла;
- Сохранение изображения в файл;
- Считывание аргументов из командной строки;
- Обработка случая некорректного ввода;
- Попиксельная обработка изображения;





СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Базовые сведения к выполнению курсовой работы по дисциплине «Программирование». Второй семестр: учеб.-метод. пособие. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2024. 36 с
2. Язык программирования Си / Б. Керниган, Д. Ритчи. Пер. с англ., 3-е изд., испр. – СПб.: «Невский диалект», 2001. 352 с: ил.
3. Репозиторий с примерами кода // moevm/pr1-examples: 1st year programming course examples. URL: <https://github.com/moevm/pr1-examples> (дата обращения: 18.04.2024)

4.

ПРИЛОЖЕНИЕ А

ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

Входные данные	Выходные данные
<code>./cw -component_min</code>	<p>=====ERROR=====</p> <p>ERROR : file must be submitted</p> <p>Command for help: —help -h</p>
<code>./cw -component_mean --input neenot.bmp</code>	<p>=====ERROR=====</p> <p>ERROR : file wasn`t read</p> <p>Command for help: —help -h</p>
<p><code>./cw -component_min -input enot.bmp</code></p> 	
<p><code>cw -component_mean -input enot.bmp</code></p> 	

ПРИЛОЖЕНИЕ В

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <getopt.h>
#include <ctype.h>
#include <string.h>
#include <math.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 freeMemory(BitmapFileHeader* bmfh, BitmapInfoHeader* bmif, Rgb
**arr){

    if (arr) {
        for (int i = 0; i < bmif->height; i++)
            free(arr[i]);
    }

    free(arr);
    free(bmfh);
```



```

        free(bmif);
    }

void panic(char* message, BitmapFileHeader* bmfh, BitmapInfoHeader* bmif,
Rgb **arr) {
    fprintf(stderr, "====ERROR====\n%sCommand for help: --help -h\n",
message);
    freeMemory(bmfh, bmif, arr);
    exit(42);
}

Rgb **read_bmp(char* file_name, BitmapFileHeader *bmfh, BitmapInfoHeader
*bmif)//чтение файла, создание экземпляров структуры Rgb.
{
    FILE *f = fopen(file_name, "rb");

    if(!f){
        panic("ERROR : file wasn't read\n", NULL, NULL, NULL);
        exit(0);
    }

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

    unsigned int H = abs((int)bmif->height);
    unsigned int W = bmif->width;

    Rgb **arr = malloc(H * sizeof(Rgb *));

    for (size_t i = 0; i < H; ++i)
    {
        arr[i] = malloc(W * sizeof(Rgb) + (4 - (W*sizeof(Rgb))%4)%4);
        fread(arr[i], 1, W * sizeof(Rgb) + (4 - (W*sizeof(Rgb))%4)%4, f);
    }

    fclose(f);
    return arr;
}

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 < abs(H); i++)
    {
        fwrite(arr[i], 1, W * sizeof(Rgb) + (4 - (W*sizeof(Rgb))%4)%4,
ff);
    }
}

```

```

        fclose(ff);
    }

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);
}

void print_help(){
    puts("Course work for option 3.5, created by Komarenko Timofei.");
    puts("ИМЯ КОМАНДЫ:");
    puts("\tИспользуйте имя ./cw, чтобы запустить программу.");
    puts("\tПо умолчанию последним аргументом утилите должно передаваться
имя входного bmp файла.");
    puts("\tИмя выходного файла по умолчанию - out.bmp, однако это можно
переопределить ключом --output/-o .");
    puts("\nСИНТАКСИС:");
    puts("\t./cw [options] [file]");
    puts("\nОПИСАНИЕ:");
    puts("\t-c, --component_mean");
    puts("\t\tУстанавливает все компоненты пикселя как среднее
арифметическое из них (округление вниз)");
    puts("\t-m, --component_min");
    puts("\t\tУстанавливает все компоненты пикселя как минимальную из
них.");
    puts("\t-i, --input");

```

```

        puts("\t\tФлаг для указания входного файла. В качестве аргумента
требуется имя файла.");
        puts("\t-o, --output");
        puts("\t\tПереопределяет имя выходного файла. В качестве аргумента
требуется имя файла.");
        puts("\t-a, --info");
        puts("\t\tВыводит информацию о BMP-файле.");
        puts("\t-h, --help");
        puts("\t\tВыводит данную справку.");
    }

void comp_mean(Rgb **arr, unsigned int H, unsigned int W)
{
    for (size_t i = 0; i < H; i++)
    {
        for (size_t j = 0; j < W; j++)
        {
            int average = (arr[i][j].r + arr[i][j].g + arr[i][j].b) / 3;

            arr[i][j].r = average;
            arr[i][j].g = average;
            arr[i][j].b = average;
        }
    }
}

void comp_min(Rgb **arr, int H, int W)
{
    for (int i = 0; i < abs(H); i++)
    {
        for (int j = 0; j < W; j++)
        {
            int min_val = arr[i][j].r;

            if(min_val > arr[i][j].g){
                min_val = arr[i][j].g;
            }
            if(min_val > arr[i][j].b){
                min_val = arr[i][j].b;
            }

            arr[i][j].r = min_val;
            arr[i][j].g = min_val;
            arr[i][j].b = min_val;
        }
    }
}

int main(int argc, char** argv){
    char* file_name = NULL;

```

```

    char* output_file = (char*)malloc((strlen("out.bmp") +
1)*sizeof(char));

    strcpy(output_file, "out.bmp");
    output_file[strlen("out.bmp")] = '\\0';

    Rgb** arr = NULL;
    BitmapFileHeader* bmfh = NULL;
    BitmapInfoHeader* bmif = NULL;

    int c = 0;
    int option_index = 0;

    static struct option long_options[] = {
        {"input", required_argument, NULL, 'i'},
        {"output", required_argument, NULL, 'o'},
        {"help", no_argument, NULL, 'h'},
        {"info", no_argument, NULL, 2},
        {"component_mean", no_argument, NULL, 3},
        {"component_min", no_argument, NULL, 4},
        {NULL, 0, NULL, 0}
    };

    c = getopt_long_only(argc, argv, "i:o:h", long_options,
&option_index);

    if(c == -1){// случай ввода некорректных опций.
        print_help();
    }

    int help_flag = 0;
    int output_flag = 0;
    const char* output_optarg = NULL;
    int mean_flag = 0;
    int min_flag = 0;
    int info_flag = 0;
    int input_flag = 0;
    const char* input_optarg = NULL;

    while (c != -1) { //работа с командной строкой.
        switch (c) {
            case 'i':
                input_flag = 1;
                input_optarg = optarg;
                break;

            case 'h':
                help_flag = 1;
                break;

            case 'o':
                output_flag = 1;
                output_optarg = optarg;
                break;

            case '?':
                printf("|%c|\\n", optopt);

```

```

        panic("ERROR : Unknown argument!\n", bmfh, bmif, arr);
        break;

    case 3:
        mean_flag = 1;
        break;

    case 4:
        min_flag = 1;
        break;

    case 2:
        info_flag = 1;
        break;

    case 5:
        gamma_flag = 1;
        break;

    case 6:
        val_optarg = atof(optarg);
        break;
    }
    c = getopt_long(argc, argv, "i:o:h", long_options,
&option_index);
}

if(input_flag){
    file_name = (char*)malloc((strlen(input_optarg)+1)*sizeof(char));
    strcpy(file_name, input_optarg);

    file_name[strlen(input_optarg)] = '\0';

    bmfh = malloc(sizeof(BitmapFileHeader));
    bmif = malloc(sizeof(BitmapInfoHeader));

    arr = read_bmp(file_name, bmfh, bmif);
}

if(help_flag){
    print_help();
    exit(0);
}

if(output_flag){
    if (output_optarg == NULL || output_optarg[0] == '-')
        panic("ERROR : no output file name!\n", bmfh, bmif, arr);
    output_file =
(char*)malloc((strlen(output_optarg)+1)*sizeof(char));

    strcpy(output_file, output_optarg);
    output_file[strlen(output_optarg)] = '\0';
}

if(mean_flag){
    comp_mean(arr, bmif->height, bmif->width);
}

```

```

    if (min_flag) {
        comp_min(arr, bmif->height, bmif->width);
    }

    if (info_flag) {
        print_info_header(*bmif);
        print_file_header(*bmfh);
    }

    if (bmif) //запись обработанного файла.
    {
        write_bmp(output_file, arr, bmif->height, bmif->width, *bmfh,
*bmf);
    }

    free(file_name);
    free(output_file);
    freeMemory(bmfh, bmif, arr);
    return 0;
}

```

Файл Makefile:

```

all:
    gcc -std=c99 main.c -lm -o cw
clean:
    rm *.o

```