**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 бинарных файлов

|  |  |  |
| --- | --- | --- |
| Студент гр. 6303 |  | Ваганов Н.А. |
| Преподаватель |  | Берленко Т.А. |

Санкт-Петербург

2017

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Ваганов Н.А. | | |
| Группа 6303 | | |
| Тема работы: | | |
|  | | |
| Содержание пояснительной записки:   * Содержание * Введение * Описание кода * Примеры работы программы * Заключение * Список использованных источников * Приложение А. Исходный код программы | | |
|  | | |
| Дата выдачи задания: 28.05.2017 | | |
| Дата сдачи: | | |
| Дата защиты: | | |
| Студент |  | Ваганов Н.А. |
| Преподаватель |  | Берленко Т.А. |

**Аннотация**

В данной работе была создана программа на языке программирования C, которая производит обработку bmp-файлов посредством набора функций. Программа считывает необходимые данные из исходного bmp-файла, инвертирует цвет в заданной области и сохраняет все данные в новый файл. В программе были использованы стандартные структуры для чтения заголовков, мета-данных и массива пикселей исходного BMP файла.

Оглавление

[Введение 1](#_Toc484128882)

[Общие сведения 1](#_Toc484128883)

[Описание кода программы 2](#_Toc484128884)

[1.1. Структура BITMAPFILEHEADER 2](#_Toc484128885)

[1.2. Структура BITMAPINFOHEADER 2](#_Toc484128886)

[1.3. Структура RGBTRIPPLE 2](#_Toc484128887)

[1.4. Структура IMAGE 3](#_Toc484128888)

[1.5 Функция main() 3](#_Toc484128889)

[Примеры работы программы 6](#_Toc484128890)

[Заключение 7](#_Toc484128891)

[Список использованных источников 8](#_Toc484128892)

[Приложение А 9](#_Toc484128893)

# Введение

Требуется написать программу, которая инвертирует цвета в заданной области BMP-файла и сохраняет результат в новом файле.

Программа получает параметры из входного потока и должна проверить их корректность. Параметры:

* input\_file
* x0
* y0
* x1
* y1
* input\_file - имя BMP файла
* x0 y0 левый верхний угол области (отсчет с точки 0, 0)
* x1 y1 правый нижний угол области

В случае, если программа получила некорректные параметры, то:

* не создается выходного в файла
* выводится сообщение об ошибке “Fail with <имя параметра>”.

### Общие сведения

* 24 бита на цвет
* без сжатия
* файл всегда соответствует формату (проверять не нужно)
* 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).

# Описание кода программы

## Структура BITMAPFILEHEADER

typedef struct BITMAPFILEHEADER

{

unsigned char bfType[2];

unsigned long bfSize;

unsigned short bfReserved1;

unsigned short bfReserved2;

unsigned long bfOffBits;

} BITMAPFILEHEADER;

Данная структура позволяет удобно считывать заголовки BMP файла для дальнейшего использования. Как и BMP файл, структура содержит 5 полей:

Формат файла (bfType), размер файла (bfSize), два зарезервированных поля (bfReserved1) и (bfReserved2) и отступ до массива пикселей (bfOffBits)

## Структура BITMAPINFOHEADER

typedef struct BITMAPINFOHEADER

{

unsigned int biSize;

int biWidth;

int biHeight;

unsigned short biPlanes;

unsigned short biBitCount;

unsigned int biCompression;

unsigned int biSizeImage;

int biXPelsPerMeter;

int biYPelsPerMeter;

unsigned int biClrUsed;

unsigned int biClrImportant;

} BITMAPINFOHEADER;

Структура BITMAPINFOHEADER хранит в себе мета-информацию о BMP файле. Структура содержит 11 полей: размер данной структуры в байтах (biSize), ширина и длина массива пикселей (соответственно biWidth и biHeight), количество плоскостей в битовом отображении (biPlanes, всегда равен 1), количество бит на один пиксель (biBitCount), метод компрессии для пикселей (biCompression), размер изображения в байтах (biSizeImage), разрешение по горизонтали и вертикали устройства вывода файла, необходимое для вывода без искажений (соответственно biXPelsPerMeter и biYPelsPerMeter), размер таблицы цветов (biClrUsed) и минимальное необходимое количество цветов для отображения файла (biClrImportant).

## Структура RGBTRIPPLE

typedef struct RGBTRIPPLE

{

unsigned char rgbBlue;

unsigned char rgbGreen;

unsigned char rgbRed;

} RGBTRIPPLE;

Для редактирования и отображения BMP файлов используется цветовая палитра RGB, в которой каждый из существующих цветов получается путем смешивания красной, зеленой и синей компоненты. Также важен и порядок расположения компонент – в BMP файлах цветовые компоненты расположены в обратном порядке: сначала синяя, потом зеленая, потом красная. Каждый пиксель изображения является элементом данной структуры.

## Структура IMAGE

typedef struct IMAGE

{

BITMAPFILEHEADER imageheader;

BITMAPINFOHEADER imageinfo;

RGBTRIPPLE \*\* imagepixels;

} IMAGE;

Структура IMAGE создана для удобства обращения к BMP файлу. Содержит перечисленные выше структуры, содержащие общую информацию о файле, мета-данные и цвет каждого элемента массива пикселей.

## Функция main()

1. Открытие файла

Для начала необходимо указать название редактируемого BMP файла и проверить, существует ли он. Если нет, то программа закрывается с сообщением об ошибке.

printf("Enter bmp file name: \n");

char \*filename=(char \*)malloc(sizeof(char)\*10);

fgets(filename, 10, stdin);

FILE \* fp = fopen("input.bmp", "rb");

if (fp == NULL)

{

printf("Error! File doesn't exist\n");

return 1;

}

1. Считывание файла

Далее мы создаем элемент структуры IMAGE, с помощью которого считываем и записываем всю необходимую информацию о файле. Для удобства обращения инициализируются переменные allx и ally, которые являются количеством строк и количеством столбцов в массиве пикселей.

Далее построчно выделяется память для создания элементов структуры RGBTRIPPLE для каждого пикселя.

Особенностью работы с BMP файлом является запись после массива пикселей т.н. «мусорных символов» для выравнивания. Программа учитывает «мусорные символы» как при считывании исходного файла, так и при записи отредактированного BMP файла. Считав всю необходимую информацию, программа закрывает исходный файл.

IMAGE \* bmp = (IMAGE\*)malloc(sizeof(IMAGE));

fread(&bmp->imageheader, sizeof(BITMAPFILEHEADER), 1, fp);

fread(&bmp->imageinfo, sizeof(BITMAPINFOHEADER), 1, fp);

int allx = bmp->imageinfo.biWidth;

int ally = bmp->imageinfo.biHeight;

RGBTRIPPLE \*\* pixel = (RGBTRIPPLE\*\*)malloc(sizeof(RGBTRIPPLE\*) \* ally);

for (i = 0; i < ally; i++)

pixel[i] = (RGBTRIPPLE\*)malloc(sizeof(RGBTRIPPLE) \* allx);

int fillingtrash = 4 - ((allx \* 3) % 4);

if (fillingtrash == 4)

{

fillingtrash = 0;

}

for (i = 0; i < ally; i++)

{

for (j = 0; j < allx; j++)

{

pixel[i][j].rgbBlue = getc(fp);

pixel[i][j].rgbGreen = getc(fp);

pixel[i][j].rgbRed = getc(fp);

}

for (k = 0; k < fillingtrash; k++)

getc(fp);

}

fclose(fp);

1. Выполнение цветовой инверсии в заданной области

После считывания исходного BMP файла программа получает на вход координаты двух точек: (x0, y0) и (x1, y1). Первая точка является левой верхней, вторая – правой нижней. Программа проверяет точки на корректность – они не должны выходить за пределы массива пикселей. В случае некорректного задания точек программа завершается, освобождая память и оставляя соответствующее сообщение об ошибке.

Если же точки заданы корректно, то программа последовательно проходит каждый пиксель в заданной области и инвертирует цвет каждой компоненты.

printf("Enter the coordinates (x0 y0 x1 y1):\n");

scanf("%d %d %d %d", &x0, &y0, &x1, &y1);

if (x0 < 0 || y0 < 0 || x1 > allx - 1 || y1 > ally - 1 || x0 > x1 || y0 > y1)

{

printf("Wrong coordinates!\n");

fclose(fp);

free(bmp);

return 1;

}

for (l = y0; l <= y1; l++)

{

for (m = x0; m <= x1; m++)

{

pixel[ally - l][m].rgbBlue = 255 - pixel[ally - l][m].rgbBlue;

pixel[ally - l][m].rgbGreen = 255 - pixel[ally - l][m].rgbGreen;

pixel[ally - l][m].rgbRed = 255 - pixel[ally - l][m].rgbRed;

}

}

bmp->imagepixels = pixel;

1. Запись нового файла

В ходе работы программа получила измененный массив пикселей, который следует сохранить как новый BMP файл. Заголовки файла и мета данные при этом не меняются, поэтому они сохраняются в исходном виде. Также вписываются необходимые байты для выравнивания.

FILE\* outputfile = fopen("output.bmp", "wb");

fwrite(&bmp->imageheader, sizeof(BITMAPFILEHEADER), 1, outputfile);

fwrite(&bmp->imageinfo, sizeof(BITMAPINFOHEADER), 1, outputfile);

for (i = 0; i < ally; i++)

{

for (j = 0; j < allx; j++)

fwrite(&bmp->imagepixels[i][j], 1, 3, outputfile);

for (k = 0; k < fillingtrash; k++)

fprintf(outputfile, "%c", 0);

}

return 0;

}

# Примеры работы программы

Пример:

Исходное изображение: Результат:

 

Инвертируем область,

ограниченную точками (150, 150), (700, 410).

Пример:

Исходное изображение: Результат:

 

Инвертируем область,

ограниченную точками (1, 1), (500, 150).

# Заключение

В ходе выполнение данной работы были изучены основы взаимодействия с бинарными файлами на примере обработки BMP файла. Были использованы стандартные структуры для считывания необходимой информации о файле. Также были изучены методы осуществления цветовой инверсии для 24-битных BMP файлов. В курсовой работе были использованы структуры, функции для работы с файлами (fread, fwrite, fclose), функции для работы с динамической памятью (malloc, free) а также стандартные методы обхода, инициализации и взаимодействия с двумерным массивом.

# Список использованных источников

1. Язык программирования СИ / Керниган Б., Ритчи Д. СПб.: Издательство "Невский Диалект", 2001. 352 с.

2. Основы программирования на языках Си и C++ [Электронный ресурс]. URL: http://cppstudio.com

# 

# Приложение А

**Исходный код программы**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#pragma pack(push)

#pragma pack(2)

typedef struct BITMAPFILEHEADER{

unsigned char bfType[2];

unsigned long bfSize;

unsigned short bfReserved1;

unsigned short bfReserved2;

unsigned long bfOffBits;

} BITMAPFILEHEADER;

typedef struct BITMAPINFOHEADER{

unsigned int biSize;

int biWidth;

int biHeight;

unsigned short biPlanes;

unsigned short biBitCount;

unsigned int biCompression;

unsigned int biSizeImage;

int biXPelsPerMeter;

int biYPelsPerMeter;

unsigned int biClrUsed;

unsigned int biClrImportant;

} BITMAPINFOHEADER;

#pragma pack(pop)

typedef struct RGBTRIPPLE{

unsigned char rgbBlue;

unsigned char rgbGreen;

unsigned char rgbRed;

} RGBTRIPPLE;

typedef struct IMAGE{

BITMAPFILEHEADER imageheader;

BITMAPINFOHEADER imageinfo;

RGBTRIPPLE \*\* imagepixels;

} IMAGE;

int main(){

int i, j, k, l = 0, m = 0;

printf("Enter bmp file name: \n");

char \*filename=(char \*)malloc(sizeof(char)\*10);

fgets(filename, 10, stdin);

FILE \* fp = fopen("input.bmp", "rb");

if (fp == NULL){

printf("Error! File doesn't exist\n");

return 1;

}

IMAGE \* bmp = (IMAGE\*)malloc(sizeof(IMAGE));

fread(&bmp->imageheader, sizeof(BITMAPFILEHEADER), 1, fp);

fread(&bmp->imageinfo, sizeof(BITMAPINFOHEADER), 1, fp);

int allx = bmp->imageinfo.biWidth;

int ally = bmp->imageinfo.biHeight;

int x0, y0, x1, y1;

printf("Enter the coordinates (x0 y0 x1 y1):\n");

scanf("%d %d %d %d", &x0, &y0, &x1, &y1);

if (x0 < 0 || y0 < 0 || x1 > allx - 1 || y1 > ally - 1 || x0 > x1 || y0 > y1){

printf("Wrong coordinates!\n");

fclose(fp);

free(bmp);

return 1;

}

RGBTRIPPLE \*\* pixel = (RGBTRIPPLE\*\*)malloc(sizeof(RGBTRIPPLE\*) \* ally);

for (i = 0; i < ally; i++)

pixel[i] = (RGBTRIPPLE\*)malloc(sizeof(RGBTRIPPLE) \* allx);

int fillingtrash = 4 - ((allx \* 3) % 4);

if (fillingtrash == 4) {

fillingtrash = 0;

}

for (i = 0; i < ally; i++) {

for (j = 0; j < allx; j++){

pixel[i][j].rgbBlue = getc(fp);

pixel[i][j].rgbGreen = getc(fp);

pixel[i][j].rgbRed = getc(fp);

}

for (k = 0; k < fillingtrash; k++)

getc(fp);

}

fclose(fp);

for (l = y0; l <= y1; l++){

for (m = x0; m <= x1; m++){

pixel[ally - l][m].rgbBlue = 255 - pixel[ally - l][m].rgbBlue;

pixel[ally - l][m].rgbGreen = 255 - pixel[ally - l][m].rgbGreen;

pixel[ally - l][m].rgbRed = 255 - pixel[ally - l][m].rgbRed;

}

}

bmp->imagepixels = pixel;

FILE\* outputfile = fopen("output.bmp", "wb");

fwrite(&bmp->imageheader, sizeof(BITMAPFILEHEADER), 1, outputfile);

fwrite(&bmp->imageinfo, sizeof(BITMAPINFOHEADER), 1, outputfile);

for (i = 0; i < ally; i++)

{

for (j = 0; j < allx; j++)

fwrite(&bmp->imagepixels[i][j], 1, 3, outputfile);

for (k = 0; k < fillingtrash; k++)

fprintf(outputfile, "%c", 0);

}

return 0;

}