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

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Структура BMP-файлов

Студент гр. 6304 _____ Курков Д. В.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург
2017

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Курков Д. В.

Группа 6304

Тема работы: структура BMP-файлов

Исходные данные:

Требуется написать программу, которая находит самый большой белый прямоугольник в BMP-файле и выводит координаты его левого верхнего и правого нижнего углов.

Содержание пояснительной записки:

Введение и постановка задачи, описание работы функции и их исходный код, результаты тестирования программы.

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Дата выдачи задания: 20.04.2017

Дата сдачи реферата: 02.06.2017

Дата защиты реферата: 02.06.2017

Студент		Курков Д. В.
Преподаватель		Берленко Т. А.

АННОТАЦИЯ

В ходе данной курсовой работы, показана реализация программы на языке программирования Си, задача которой найти наибольший прямоугольник белого цвета в заданном BMP-файле.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной курсовой работы является получение практических навыков по обработки файлов средствами, предоставляемыми языком Си, на примере работы с BMP-файлами.

Задачи:

- Изучение структуры строения BMP-файлов
- Создание работоспособной программы, отвечающей заданию
- Тестирование созданной программы

СТРУКТУРА BMP-ФАЙЛА

Указана структура для файлов, BMP-файлов, которые используются в данной работе, то есть для файлов.

Bitmapheader

Поз. (hex)	Размер (байты)	Имя	Тип WinAPI	Описание
00	2	bfType	WORD	Отметка для отличия формата от других (сигнатура формата). Может содержать единственное значение $4D42_{16}$/$424D_{16}$ (little-endian/big-endian).
02	4	bfSize	DWORD	Размер файла в байтах.
06	2	bfReserved1	WORD	Зарезервированы и должны содержать ноль.
08	2	bfReserved2	WORD	
0A	4	bfOffBits	DWORD	Положение пиксельных данных относительно начала данной структуры (в байтах).

Bitmapinfo (представленна версия в 32 бит, актуальная для данной)

Позиция в файле (hex)	Позиция в структуре (hex)	Размер (байты)	Тип WinAPI	Описание
0E	00	4	DWORD	Размер данной структуры в байтах, указывающий также на версию структуры (см. таблицу версий выше).
12	04	4	LONG	Ширина растра в пикселях. Указывается целым числом со знаком. Ноль и отрицательные не документированы.
16	08	4	LONG	Целое число со знаком, содержащее два параметра: высота растра в пикселях (абсолютное значение числа) и порядок следования строк в двумерных массивах (знак числа). Нулевое значение не документировано.
1A	0C	2	WORD	В BMP допустимо только значение 1. Это поле используется в значках и курсорах Windows.
1C	0E	2	WORD	Количество бит на пиксель
1E	10	4	DWORD	Указывает на способ хранения пикселей.
22	14	4	DWORD	Размер пиксельных данных в байтах.
26	18	4	LONG	Количество пикселей на метр по горизонтали и вертикали
2A	1C	4	LONG	
2E	20	4	DWORD	Размер таблицы цветов в ячейках.
32	24	4	DWORD	Количество ячеек от начала таблицы цветов до последней используемой (включая её саму).

ПОСТАНОВКА ЗАДАЧИ

Требуется написать программу, которая находит самый большой белый прямоугольник в BMP-файле и выводит координаты левого верхнего и правого нижнего его углов.
Программа получает параметры их входного потока и должна проверить их корректность.
Параметр:

- input_file — имя BMP файла

В случае, если программа получила некорректный параметр, то:

- выводится сообщение об ошибке «Fail».

Общие сведения:

- 24 бита на цвет
- без сжатия
- файл всегда соответсвует формату

2. РЕАЛИЗАЦИЯ ПРОГРАММЫ

ХРАНЕНИЕ ПРОМЕЖУТОЧНЫХ РЕЗУЛЬТАТОВ

Для хранения промежуточных результатов создана структура **rect,** содержащая характеристику найденных прямоугольников(координаты его левого нижнего и правого верхнего угла, и его площадь), также созданы два массива для хранения правильных **arr** и неправильных **r_arr** прямоугольников, неправильные прямоугольники используются для проверки, осуществляемой функцией **is_rect.**
Для работы с вышеописанными массивами реализованы функции **push**:

```
void push (rect A, char m)
```

и **rmv**:

```
void rmv (int n).
```

push определяет полученный прямоугольник A в массив, согласно переменной **m, m** может принимать значения «**r**», *поместить в массив* **r_arr** или «**a**», *поместить в массив* **arr**.

rmv удаляет элемент из массива правильных прямоугольников, если он был ложно ошибочно принят за правильный.

P. S. Исходный код всех функций, описанных в пункте 2, можно найти в приложении.

ПОИСК В BMP-ФАЙЛЕ

Для поиска правильного прямоугольника в BMP-файле реализованы две функции **rect_searching**, построчно обрабатывающая байты BMP-файла и ищущая последовательность белых пикселей, и **is_rect,** проверяющая найденные прямоугольники.

- **rect_searching**

  Прототип функции выглядит следующим образом:

  void rect_searching (FILE *file, **int** widht, **int** height, **int** padding).

  Функция получает из **main** указательна файл, его ширину и высоту в пикселях, а также количество «мусорных» байт. Используемых для выравнивания. После чего **rect_searching** построчно обрабатывает пиксели BMP-файла, и находит все последовательности белых пикселей в строке, после чего передает каждую функции is_rect, в формате rect.

- **is_rect**

  void is_rect (rect A).

  Функция принимает объект структуры rect, после чего проверяет отвечает ли этот объект условиям задания, если объект подходящий функция помещает его в массив, где содержаться все подходящие прямоугольники, в случае же если объект не подходит под условия задания, он помещается в массив для неправильных прямоугольников.

ФУНКЦИЯ MAIN

Содержит основную функцию main, прототип этой функции выглядит следующим образом:

int *main* ***(int*** *count*, ***char*** ** *input*).

Предназначение этой функции считать данные от пользователя, проверить их на корректность, после чего в случае успешной проверки вызвать необходимые функции для работы с полученными данными и обработать возвращаемые ими значения, после чего передать пользователю результат, в противном случае вывести сообщение об ошибке и завершить программу.

В качестве аргумента функции принимается массив строк, при корректном пользовании программой, в массиве должна содержаться лишь одна строка — название BMP-файла (то есть переменная count = 1).

ТЕСТИРОВАНИЕ ПРОГРАММЫ

Тестирование происходит путем подачи BMP-файлов на вход программе. Все BMP-файлы можно отыскать в приложении номер теста совпадает с номером файла в приложении.

Input	**Output**
Hello my name is Denis	**Error, wrong input**
Test_1.bmp Изображение 100x100 пикселей, представляющие собой белый прямоугольник на черной фоне	Top left corner: X1 = 18 Y1 = 47 Bottom right corner: X2 = 83 Y2 = 10
Test_2.bmp Изображение 465x385 пикселей, изображающей просто черный квадрат.	There are no white rectangles in ./Test_2.bmp
Test_3.bmp Изображение 1x1 пиксель, просто белый пиксель.	Top left corner: X1 = 0 Y1 = 0 Bottom right corner: X2 = 0 Y2 = 0
Test_4.bmp Изображение 465x385 пикселей, три наложенных друг на друга прямоугольниика.	There are no white rectangles in ./Test_4.bmp
Test_5.bmp Изображение 1000x1000 пикселей, представляет собой хаотично разбросанные по цветному фону белые прямоугольники.	Top left corner: X1 = 781 Y1 = 518 Bottom right corner: X2 = 999 Y2 = 84
Nya.bmp	Top left corner: X1 = 432 Y1 = 147 Bottom right corner: X2 = 445 Y2 = 135
black_square.bmp Поднимает культурное значение этой работы.	Top left corner: X1 = 384 Y1 = 47 Bottom right corner: X2 = 392 Y2 = 43

РАЗМЕЩЕНИЕ РАБОТЫ В РЕПОЗИТОРИИ ГРУППЫ

1. Загружаем клон репозитория на компьютер, для этого необходимо выполнить команду
git clone https://github.com/moevm/pr1-2016-6304.git

2. Теперь необходимо создать ветку для последующего сохранения туда программы, делается это следующей командой
git checkout -b KURKOV_COURSEWORK,

создана ветка KURKOV_COURSEWORK.

3. Перемещаем папку с программой в эту ветку.

4. Добавим информацию об изменения командой *git add KURSWORK*, где KURSWORK — название папки с программой.

5. Подтвердим изменения командой git commit -m «kurswork added».

6. Сохраним изменения командой *git push origin KURKOV_KURSWORK*, необходимо ввести свой логин и пароль

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Язык программирования СИ / Керниган Б., Ритчи Д. Издательский дом «Вильямс» 2016 г.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были выполнены все поставленные задачи: изучено строение BMP-файлов, создана и протестирована программа для поиска наибольшего прямоугольника белого цвета в заданном BMP-Файле.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>

//Структура, представляющая прямоугольник
typedef struct rect
{
	int x1, x2, y1, y2;
	unsigned int square;
}rect;

//Массив для правильных прямоугольников
rect arr[10000];
int sz = 0;

//Массив для неправильных прямуогольников
rect r_arr[10000];
int r_sz = 0;

//Функция для добавления в массив: a - правильный, r - неправильный
void push (rect A, char m)
{
	switch (m)
	{
		case 'a':
			*(arr+(sz++)) = A;
			break;
		case 'r':
			*(r_arr+(r_sz++)) = A;
			break;
	}
}

//Удаление элемента из массива "правильного" массива
void rmv (int n)
{
	for (int i = n, j = n+1; j < sz; i++, j++)
		*(arr+i) = *(arr+j);
	sz--;
}

//Проверка прямоугольника-кандидата
void is_rect (rect A)
{
	int i, f = 0;
	for (i = 0; i < sz ; i++)
	{
		//При совпадении координат, увеличиватся площадь
```

```
            if (((arr+i)->x1 == A.x1) && ((arr+i)->x2 == A.x2) && ((arr+i)->y2 == (A.y1-1)))
            {
                (arr+i)->square += A.square;
                (arr+i)->y2 = A.y1;
                f = 1;
            }
            //Если прямоугольник не правильный, он удаляется
            else if (((((arr+i)->x1 >= A.x1) && ((arr+i)->x1 <= A.x2)) ||
                        (((arr+i)->x1 <= A.x1) && ((arr+i)->x2 >= A.x1)))
                        && ((arr+i)->y2 == (A.y1 - 1)))
            {
                rmv(i--);
                f = 1;
            }

        }
        if (!f)
        {
            //Проверка на правильность
            for (i = 0; i < r_sz; i++)
            {
                if (((((r_arr+i)->x1 >= A.x1) && ((r_arr+i)->x1 <= A.x2)) ||
                    (((r_arr+i)->x1 <= A.x1) && ((r_arr+i)->x2 >= A.x1)))
                    && ((r_arr+i)->y2 == (A.y1 - 1)))
                    break;
            }
            if (i == r_sz)
                push(A,'a');
            else
                push (A,'r');
        }
        else
            push(A, 'r');
}
//Поиск последовательности белых пиксилей
void rect_searching (FILE *file, int widht, int height, int padding)
{
    rect A;
    int pixel[3], mod = 0;
    fseek(file, 54, SEEK_SET);
    for (int y = 0; y < height; y++)
    {
        for (int x = 0; x < widht-2; x += 3)
        {
            for (int i = 0; i < 3; i++)
                pixel[i] = fgetc(file);
            if ((pixel[0] != 0xff || pixel[1] != 0xff || pixel[2] != 0xff))
            {
                if (mod)
                {
                    mod = 0;
                    A.x2 = (x-3)/3;
```

```
                        A.square = A.x2 - A.x1 + 1;
                        is_rect(A);
                    }
                    continue;
                }
                if (!mod)
                {
                    A.x1 = x/3;
                    A.y1 = A.y2 = y;
                    mod = 1;
                }
                if (mod && x == widht-3)
                {
                    A.x2 = x/3;
                    A.square = A.x2 - A.x1 + 1;
                    is_rect(A);
                    mod = 0;
                }
            }
            //"Мусорные" байты
            fseek(file, padding, SEEK_CUR);
        }
}

int main (int count, char** input)
{
        FILE *bmp = fopen(input[1], "r");
        if (bmp == NULL || !(strstr(input[1], ".bmp")))
        {           printf ("Error, wrong input\n");
                    return 0;
        }
        //Вычисляем ширину изображения
        fseek(bmp, 18, SEEK_SET);
        int widht = 0;
        for (int i = 0; i < 4; i++)
            widht += fgetc(bmp)*pow(256, i);
        //Вычисляем высоту изображения
        int height = 0;
        for (int i = 0; i < 4; i++)
            height += fgetc(bmp)*pow(256, i);
        int b_widht = ((widht*3)%4) ? ((widht*3)/4+1)*4 : widht*3; //Длина строки в байтах
        int ix = 0;
        rect_searching(bmp, widht*3, height, b_widht-widht*3);
        if (!sz)
            printf ("There are no white rectangles in %s\n", input[1]);
        else
        {
            for (int i = 0; i < sz; i++)
                ix = ((arr+i)->square > (arr+ix)->square) ? i : ix;
            printf ("Top left corner:\n\t\tX1 = %d\n\t\tY1 = %d\nBottom right corner:\n\t\tX2 =
%d\n\t\tY2 = %d\n", (arr+ix)->x1, (arr+ix)->y2, (arr+ix)->x2, (arr+ix)->y1);
        }
```

```
    fclose(bmp);
    return 0;
}
```