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 работы: 28.06.2017

Дата защиты работы: 28.06.2017

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 с 32-битным информационным полем.

Bitmapheader

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

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

Позиция в файле (hex)	Позиция в структуре (hex)	Размер (байты)	Описание
0E	00	4	Размер данной структуры в байтах, указывающий также на версию структуры (см. таблицу версий выше).
12	04	4	Ширина растра в пикселях. Указывается целым числом со знаком. Ноль и отрицательные не документированы.
16	08	4	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	В BMP допустимо только значение 1. Это поле используется в значках и курсорах Windows.
1C	0E	2	Количество бит на пиксель
1E	10	4	Указывает на способ хранения пикселей.
22	14	4	Размер пиксельных данных в байтах.
26	18	4	Количество пикселей на метр по горизонтали и вертикали
2A	1C	4	
2E	20	4	Размер таблицы цветов в ячейках.
32	24	4	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

РЕАЛИЗАЦИЯ ПРОГРАММЫ

СТРУКТУРА ПРОГРАММЫ

Программа состоит из трех файлов с исходным кодом: get_matrix.c, searching.c и main.c. Для работы функций описанных в get_matrix и main.c требуется структура rectangle, она содердится в файле rectangle.h.

rectangle.h:

```
#include <stdio.h>

typedef struct rectangle
{
	int x1, x2, y1, y2;
	int square;
}rectangle;
```

GET_MATRIX.C

В этом файле содержится функция

```
int *get_matrix(char *file, int* height, int* widht),
```

преобразующая bmp-файл в целочисленный массив, который в дальнейшем будет взаимодействовать программа.

В качестве аргументов в get_matrix подаются строка с именем файла и два указателя на int, которые примут значения высоты и ширины считанной матрицы соответственно.

При удачном открытии файла функция get_matrix возвращает указатель на int, указывающий на матрицу, а в противном случае NULL.

Считывание матрицы происходит следующим образом:

- Сначала функция получает из информационного поля bmp-файла значения его высоты и ширины, и основываясь на полученных данных вычисляет padding("мусорные" байты).
- Далее в функции выделяется память для хранения будущей матрицы.
- После чего, происходит считывание пикселей изображения, каждому пикселю соответствует в матрице соответствует 0 или 1, 1 обозначает белый цвет, а 0 любой другой.
- Считанная, матрицы возвращается в вызывающую функцию main.

SEARCHING.C

В этом файле содержатся функции rect_search, ищущая прямоугольные области, состоящие из одних единиц, и is_rect, проверяющая найденные в rect_search прямоугольники, наибольший прямоугольник возвращается в main.

Принцип работы rect_search состоит в следующем: построчно обрабатывая элементы матрицы, для каждого элемента строки находятся:

a. нижняя граница, то есть ближайший снизу нуль в столбце, номер строки, в которой находится этот нуль заносится в специальный массив b[M].
b. левая граница элемента, ближайшей левый столбец элемента x1, b[x1] которого больше чем, b[x] самого столбца. Для этого используется стек, в вершине стека всегда находится элемент находящийся слева от x, элементы извлекаются из стека до тех пор пока b[x1] <= b[x], если нужный элемент не найден, а стек пуст, то левая граница данного элемента — левая граница изображения.
c. Аналогично находится правая граница.

После чего полученные данные используются для того, чтобы вычислить координаты прямоугольника-кандидата, и его площадь. Если площадь найденного прямоугольника больше площади прямоугольника result и функция is_rect возвращает положительное значений, result присваивается значение найденного прямоугольника. После того, как все элементы матрицы просмотрены функция rect_search возвращает значение result в main.

Функция is_rect проверяет, является ли переданный ей из rect_search прямоугольник корректным, для этого все элементы окружающие прямоугольник проверяются на неравество единице (белому цвету).

MAIN.C

Функция main занимается обработкой результатов работы вышеописанных функций.

- В случае если, что-то пошло не так при открытии файла, функция выводит «*Error, wrong input*».
- Если в bmp-файле отсутствуют белые прямоугольники, функция выводит «There are no white rectangles in "file_name"» .
- Если bmp-файл содержит белые прямоугольники, то функция выводит координаты левого верхнего и правого нижнего угла наибольшего из них в формате:

Top left corner:
x = ?
y = ?
Bottom right corner:
x = ?
y = ?

ТЕСТИРОВАНИЕ ПРОГРАММЫ

Тестирование происходит путем подачи BMP-файлов на вход программе. Все BMP-файлы можно отыскать в директории с работой.

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 = 82 Y2 = 10
Test_2.bmp Изображение 465x385 пикселей, только черный цвет.	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 прямоугольника.	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	Top left corner: X1 = 384 Y1 = 40 Bottom right corner: X2 = 387 Y2 = 38

СОЗДАНИЕ ИСПОЛНЯЕМОГО ФАЙЛА

Для того чтобы получить из файлов, содержащих исходный код, исполняемый файл, используется утилита make, которой подается на вход текстовый файл следующего содержания.

```
all: programm
	rm *.o

programm: main.o get_matrix.o searching.o
	gcc -o programm main.o get_matrix.o searching.o -lm

main.o: main.c
	gcc -c main.c

get_matrix.o: get_matrix.c
	gcc -c get_matrix.c

searching.o: searching.c
	gcc -c searching.c
```

РАЗМЕЩЕНИЕ РАБОТЫ В РЕПОЗИТОРИИ ГРУППЫ

1. Загружаем клон репозитория на компьютер, командой

git clone https://github.com/moevm/pr1-2016-6304.git

2. Командой

git checkout -b KURKOV_COURSEWORK,

в репозитории создается новая ветку, куда следует поместить работу.

4. Информация об изменениях добавляется командой *git add COURSEWORK*, где COURSEWORK — название папки с программой.

5. Изменения подтверждаются командой git commit -m «coursework added».

6. Чтобы изменения сохранились, используется команда *git push origin KURKOV_COURSEWORK*, необходимо ввести свой логин и пароль.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Язык программирования СИ / Керниган Б., Ритчи Д. Издательский дом «Вильямс» 2016 г.
2. Википедия — свободная энциклопедия //Wikipedia. URL: https://ru.wikipedia.org/wiki/BMP
3. MAXimal — веб-сайт//URL: http://www.e-maxx-ru.1gb.ru/algo/maximum_zero_submatrix

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

rectangle.h:

```
#include <stdio.h>

typedef struct rectangle
{
	int x1, x2, y1, y2;
	int square;
}rectangle;
```

get_matrix.h:

```
int *get_matrix (char *, int *, int *);
```

searching.h:

```
rectangle rect_search(int *, int , int );
```

get_matrix.c

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

int *get_matrix(char *file, int* height, int* widht)
{
	FILE *bmp = fopen(file, "r");
	if (!bmp)
		return NULL;
	fseek(bmp, 18, SEEK_SET);
	for (int i = 0; i < 4; i++)
		*widht += fgetc(bmp) * pow(256, i);
	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;
	int padding = b_widht - *widht*3;
	int *matrix = (int*)malloc(sizeof(int)*(*height)*(*widht));
	int pxl[3];
	fseek(bmp, 54, SEEK_SET);
	for (int y = 0; y < *height; y++)
	{
		for (int x = 0; x < *widht; x++)
		{
			for (int i = 0; i < 3; i++)
				pxl[i] = fgetc(bmp);
			*(matrix+y*(*widht)+x) = (pxl[0] != 0xff || pxl[1] != 0xff || pxl[2] != 0xff) ? 0 : 1;
		}
		fseek(bmp, padding, SEEK_CUR);
	}
	fclose(bmp);

	return matrix;
}
```

searching.h

```
#include "rectangle.h"

int is_rect(rectangle rect, int *matrix, int N, int M)
{
	int x, y;
	for (y = rect.y1, x = rect.x1-1; y <= rect.y2 && x > 0;  y++)
		if (*(matrix+y*M+x) != 0)
			return 0;
	for (y = rect.y2, x = rect.x2+1; y >= rect.y1 && x < M; y--)
		if(*(matrix+y*M+x) != 0)
			return 0;
	for (y = rect.y1-1, x = rect.x1; x <= rect.x2 && y >= 0; x++)
		if(*(matrix+y*M+x) != 0)
			return 0;
	for (y = rect.y2+1, x = rect.x2+1; x >= rect.x1 && y < N; x--)
		if(*(matrix+y*M+x) != 0)
			return 0;
	return 1;
}

rectangle rect_search(int *matrix, int N, int  M )
{
	rectangle result;
	result.square = 0;
	int b[M], r[M], l[M];
	for (int i = 0; i < M; i++) {
		b[i] = l[i] = -1;
		r[i] = M;
	}
	int st[M], sz = 0;
	for (int y = 0; y < N; y++)
	{
		for (int x = 0; x < M; x++)
			if (*(matrix+y*M+x) == 0)
				b[x] = y;
		while (sz > 0) sz--;
		for (int x = 0; x < M; x++){
			while(sz > 0 && (b[st[sz]] <= b[x])) sz--;
			l[x] = (sz) ? st[sz] : -1;
			st[++sz] = x;
		}
		while (sz >= 0) sz--;
		for (int x = M-1; x >= 0; x--){
			while (sz > 0 && (b[st[sz]] <= b[x])) sz--;
			r[x] = (sz >= 0) ? st[sz] : M;
			st[++sz] = x;

		}
		for (int x = 0; x < M; x++){
			rectangle t_rect;
			t_rect.x1 = l[x]+1; t_rect.x2 = r[x]-1;
			t_rect.y1 = b[x]+1; t_rect.y2 = y;
			t_rect.square = (r[x]-l[x]-1)*(y-b[x]);
			if (t_rect.square > result.square)
				result = (is_rect(t_rect, matrix, N, M)) ? t_rect : result;
		}
	}
	return result;
}
```

main.c

```c
#include <stdlib.h>
#include "rectangle.h"
#include "get_matrix.h"
#include "searching.h"

int main (int count, char** input)
{
	int height = 0, widht = 0;
	int *matrix = get_matrix(input[1], &height, &widht);
	if (!matrix)
	{
		printf ("Error, wrong input!\n");
		return 0;
	}
	rectangle answer = rect_search(matrix, height, widht);
	if(!answer.square)
		printf ("There are no white rectangles in %s\n", input[1]);
	else
		printf ("Top left corner:\n\t\t%d\n\t\t%d\nBottom right corner\n\t\t%d\n\t\t%d\n", answer.x1,
answer.y2, answer.x2, answer.y1);
	free(matrix);
	return 0;
}
```