**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. 6304 |  | Попов С.С. |
| Преподаватель |  | Берленко Т.А. |

Санкт-Петербург

2017

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студент Попов С.С. | | |
| Группа 6304 | | |
| Тема работы: Обработка BMP- файла | | |
|  | | |
| Содержание пояснительной записки:   * Содержание * Введение * Описание функций, необходимых для работы программы * Примеры работы программы * Заключение * Список использованных источников * Приложение А. Исходный код программы | | |
| Объем пояснительной записки: 30 страниц. | | |
| Дата выдачи задания: 28.04.2017 | | |
| Дата сдачи: | | |
| Дата защиты: | | |
| Студент |  | Попов С.С. |
| Преподаватель |  | Берленко Т.А. |

**Аннотация**

В данной курсовой работе была реализована программа на языке C, которая производит чтение, изменение и сохранение BMP-файла. Были реализованы и описаны функции, необходимые для открытия, разбора и редактирования файла, а также для создания и сохранения измненений в новый файл. Также присутствует описание работы с утилитой make, необходимой для удобного разбиения проекта; описание тестирования работоспособности программы и скриншоты с полученными изображениями.

Содержание

[Введение 4](#_Toc484361348)

[Цель работы 4](#_Toc484361349)

[Формулировка задачи 4](#_Toc484361350)

[Параметры: 4](#_Toc484361351)

[Общие сведения 5](#_Toc484361352)

[1. Теоретические сведения 6](#_Toc484361353)

[1.1 Структура BITMAPFILEHEADER 8](#_Toc484361354)

[1.2 Структура BITMAPINFOHEADER 8](#_Toc484361355)

[2. Функции для работы с BMP-файлом 9](#_Toc484361356)

[2.1 Считывание BMP-файла 9](#_Toc484361357)

[2.2 Изменение BMP-файла 10](#_Toc484361358)

[2.3 Сохранение измененного BMP-файла в новый файл 11](#_Toc484361359)

[2.4 Функция main 12](#_Toc484361360)

[Считывание входных данных 12](#_Toc484361361)

[Проверка заданной области на валидность 12](#_Toc484361362)

[Считывание файла по его названию 12](#_Toc484361363)

[Переопределение исходных данных 13](#_Toc484361364)

[Проверка совместимости размеров заданной области и файла 13](#_Toc484361365)

[Вызов функции для закрашивания файла, кроме заданной области 13](#_Toc484361366)

[Сохранения результата в новый файл 13](#_Toc484361367)

[3 Создание makefile 13](#_Toc484361368)

[4 Примеры работы программы 14](#_Toc484361369)

[Заключение 18](#_Toc484361370)

[Список использованной литературы 19](#_Toc484361371)

[Приложение А. Исходный код программы 20](#_Toc484361372)

# Введение

## Цель работы

Изучение структуры и способов обработки BMP-файла. Создание программы, отвечающей условиям поставленной задачи.

## Формулировка задачи

Требуется написать программу, которая заполняет пространство вне заданной квадратной области BMP-файла черным цветом 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 заданная область не является квадратной, то:

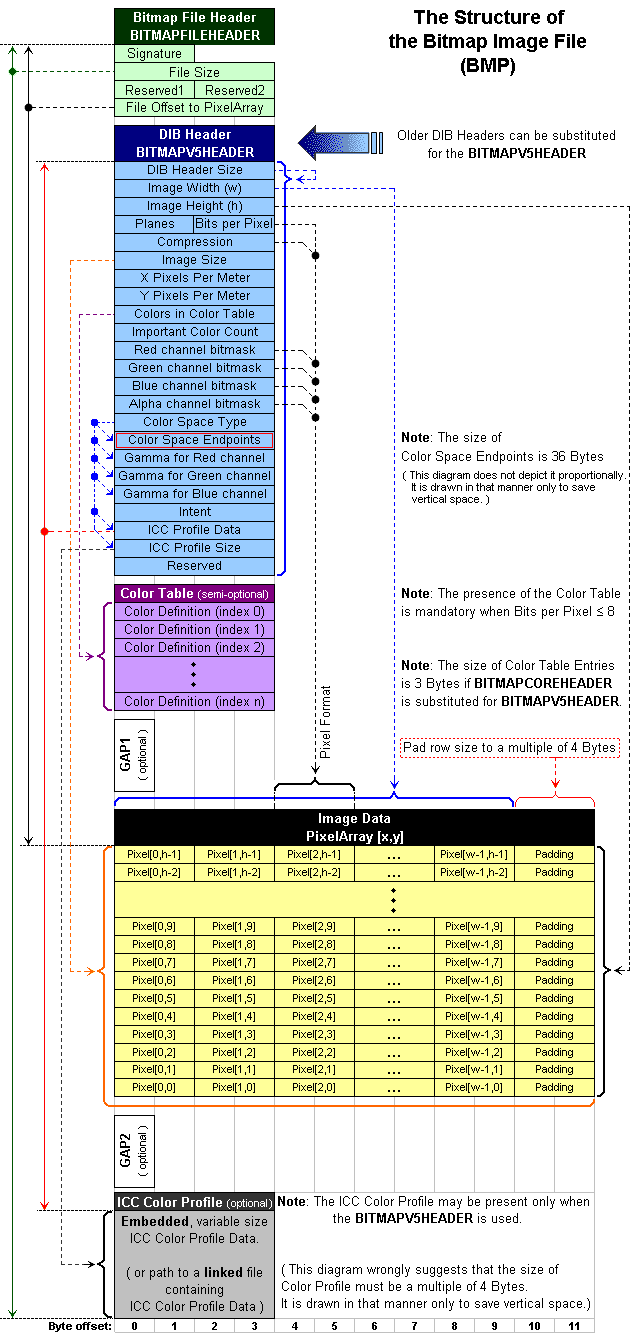
* Не создается выходного файла
* Программа должна вывести строку “The area isn't correctly defined”.

### Общие сведения

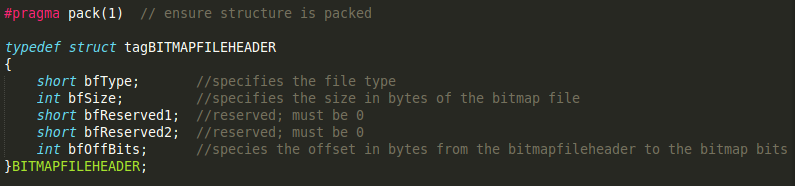
1. 24 бита на цвет
2. Без сжатия
3. Файл всегда соответствует формату (проверять не нужно)
4. Обратите внимание на выравнивание; мусорные данные, если их необходимо дописать в файл для выравнивания, должны быть нулями.
5. Обратите внимание на порядок записи пикселей
6. Все поля стандартных BMP заголовков в выходном файле должны иметь те же значения что и во входном (разумеется кроме тех, которые должны быть изменены).

# Теоретические сведения

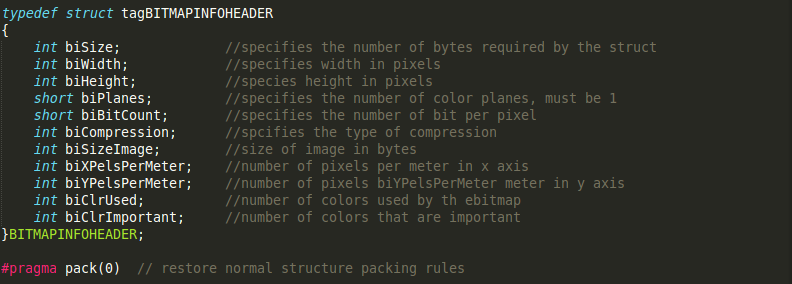
BMP (от англ. Bitmap Picture) — формат хранения растровых изображений, разработанный компанией Microsoft. Файлы формата BMP могут иметь расширения .bmp, .dib и .rle.  
Данные в формате BMP состоят из трёх основных блоков различного размера:

1. Заголовок из структуры [BITMAPFILEHEADER](https://ru.wikipedia.org/wiki/BMP#BITMAPFILEHEADER) и блока [BITMAPINFO](https://ru.wikipedia.org/wiki/BMP#BITMAPINFO). Последний содержит:
   1. Информационные поля.
   2. [Битовые маски для извлечения значений цветовых каналов](https://ru.wikipedia.org/wiki/BMP#.D0.9C.D0.B0.D1.81.D0.BA.D0.B8_.D0.BA.D0.B0.D0.BD.D0.B0.D0.BB.D0.BE.D0.B2) (опциональные).
   3. [Таблица цветов](https://ru.wikipedia.org/wiki/BMP#.D0.A2.D0.B0.D0.B1.D0.BB.D0.B8.D1.86.D0.B0_.D1.86.D0.B2.D0.B5.D1.82.D0.BE.D0.B2) (опциональная).
2. Цветовой профиль (опциональный).
3. [Пиксельные данные](https://ru.wikipedia.org/wiki/BMP#.D0.9F.D0.B8.D0.BA.D1.81.D0.B5.D0.BB.D1.8C.D0.BD.D1.8B.D0.B5_.D0.B4.D0.B0.D0.BD.D0.BD.D1.8B.D0.B5).
4. 

## Структура BITMAPFILEHEADER



## Структура BITMAPINFOHEADER



Структура BITMAPFILEHEADER должна занимать 14 байт, а BITMAPInfoHeader – 40, но из-за выравнивания эти структуры занимают большее количество байт. Для решения данной проблемы необходимо отменить выравнивание с помощью директивы #pragma.

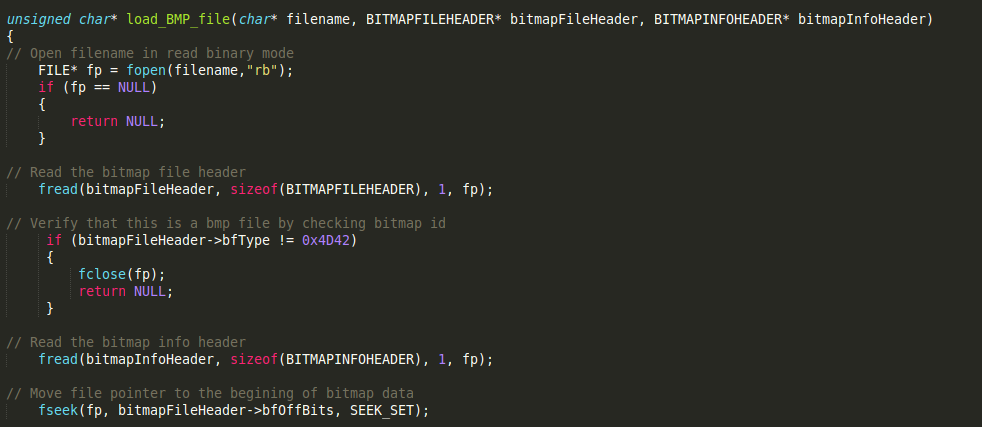
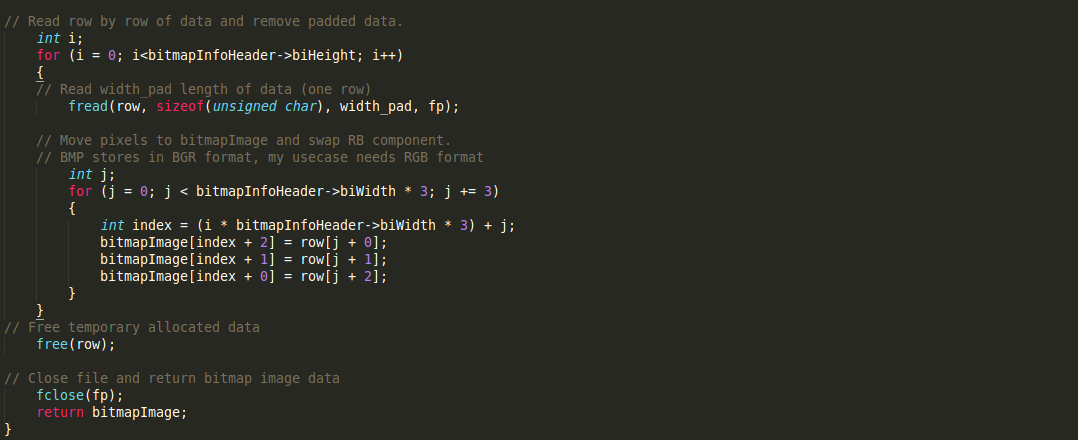
#pragma – определяемая реализацией директива, которая позволяет передавать компилятору различные инструкции.

#pragma pack(1) - устанавливает выравнивание в 1 байт

#pragma pack(0) – восстанавливает стандартные правила выравнивания

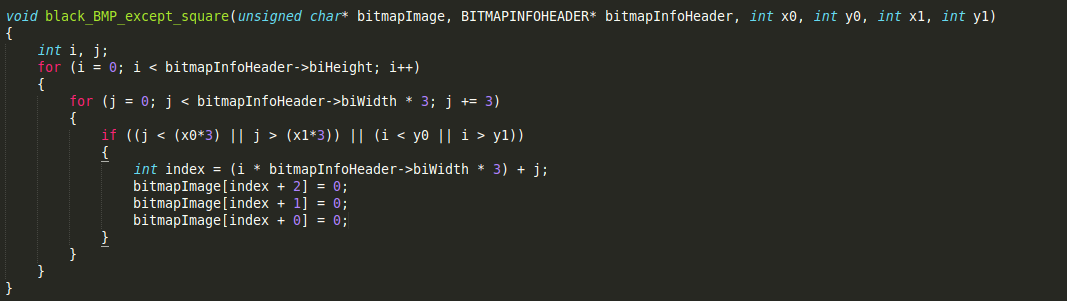
# Функции для работы с BMP-файлом

## Считывание BMP-файла

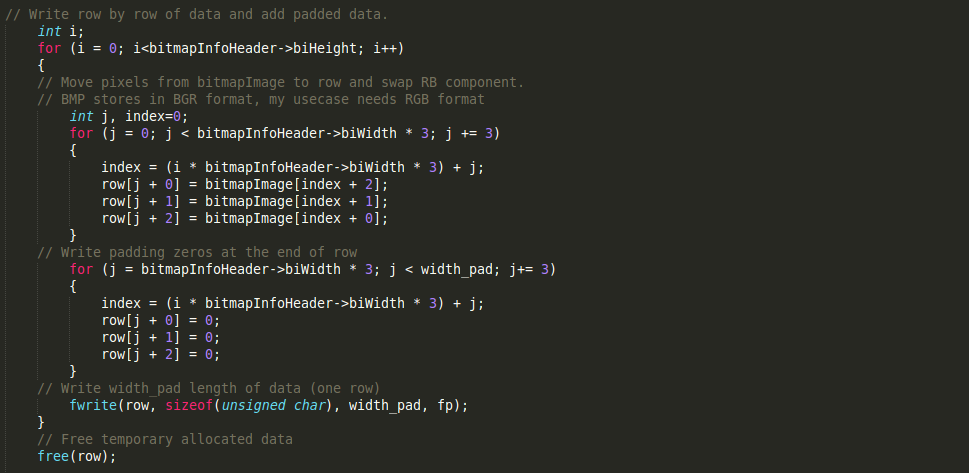
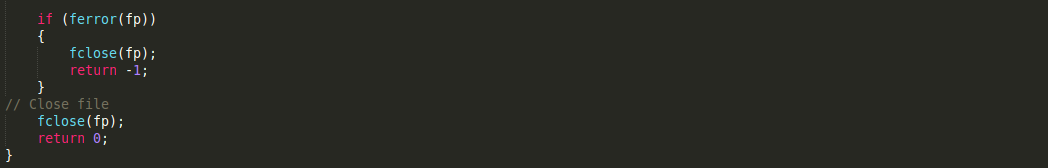
Функция принимает в качестве аргументов название файла (char\* filename), а также указатели на пустые структуры BITMAPFILEHEADER (BITMAPFILEHEADER\* bitmapFileHeader) и BITMAPINFOHEADER(BITMAPINFOHEADER\* bitmapInfoHeader) и возвращает массив пикселей самого изображение, извлеченного из файла.

## Изменение BMP-файла



Функция принимает в качестве аргументов указатель на массив пикселей (unsigned char\* bitmapImage), указатель на структуру BITMAPINFOHEADER (BITMAPINFOHEADER\* bitmapInfoHeader), а также координаты точек, предположительно задающих квадратную область внутри изображения (int x0, int y0, int x1, int y1). Изменяет массив пикселей по указателю, следовательно, отсутствует необходимость что-либо возвращать.

## Сохранение измененного BMP-файла в новый файл

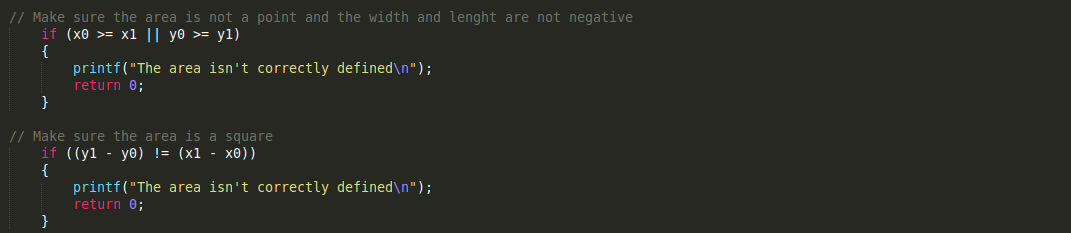
Функция принимает в качестве аргументов имя нового файла(char\* filename), указатели на структуры BITMAPFILEHEADER (BITMAPFILEHEADER\* bitmapFileHeader) и BITMAPINFOHEADER (BITMAPINFOHEADER\* bitmapInfoHeader), а также указатель на массив пикселей, составляющих изображение (unsigned char\* bitmapImage) и возвращает 0 в случае успешного сохранения файла, и -1 в случае ошибки.

## Функция main

### Считывание входных данных

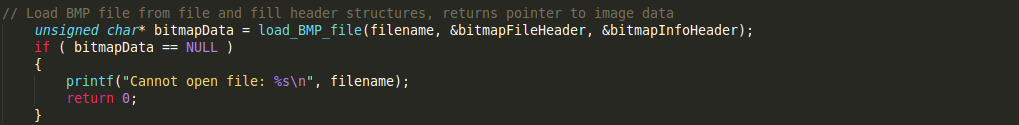


### Проверка заданной области на валидность

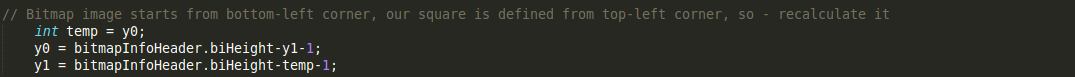


Первое условие проверяет, что заданная область не является точкой, а также, что длина и ширина больше нуля. Второе условие проверяет, что заданная область действительно является квадратом.

### Считывание файла по его названию

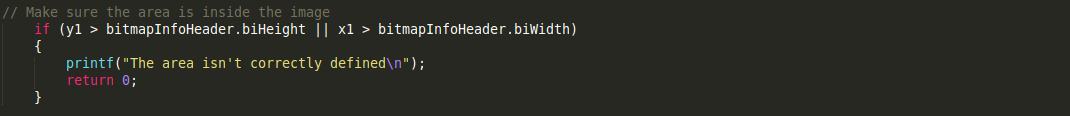


### Переопределение исходных данных



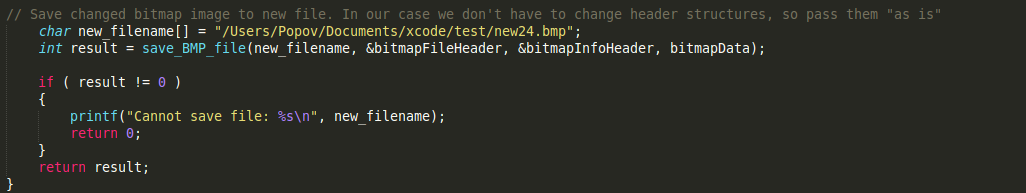
Так как отсчет изображения файла начинается с левого нижнего угла, а заданная область определена с верхнего левого угла, необходимо произвести пересчет исходных данных.

### Проверка совместимости размеров заданной области и файла

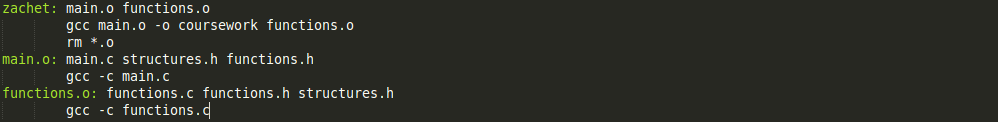


Вызов функции для закрашивания файла, кроме заданной области  


### Сохранения результата в новый файл



# Создание makefile



# Примеры работы программы

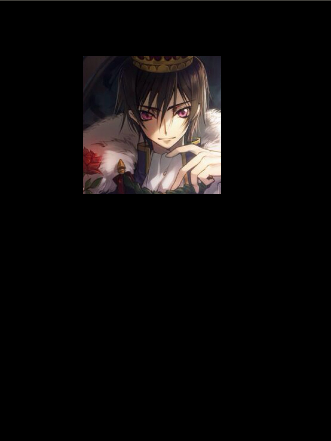
**Пример 1:**

**Начальное изображение**

**Входные данные**



**Результат**



**Пример 2:**

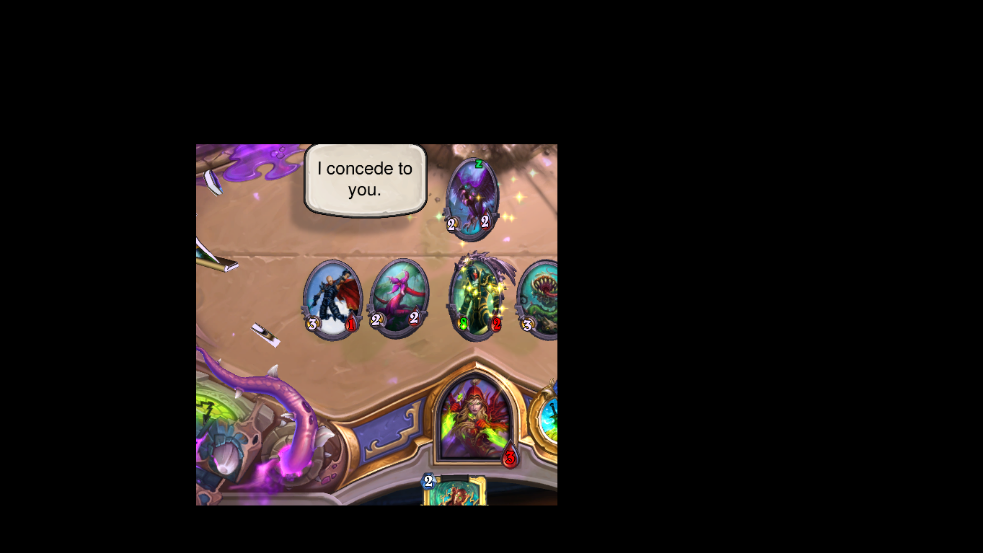
**Начальное изображение**



**Входные данные**



**Результат**



**Пример 3:**

**Начальное изображение**



**Входные данные**



**Результат**



# Заключение

В ходе данной курсовой работы была создана программа на языке программирования C, которая позволяет считывать, обрабатывать и сохранять BMP-файлы. Были на практике закреплены навыки работы с файлами такого формата. Были приведены примеры работы программы как с успешным завершением и созданием нового файла, так и с досрочным завершением работы программы в связи с невалидностью входных данных.

# Список использованной литературы

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

# Приложение А. Исходный код программы

**kursa4.c**

#include <stdio.h>

#include <stdlib.h>

#pragma pack(1) // ensure structure is packed

typedef struct tagBITMAPFILEHEADER

{

short bfType; // specifies the file type

int bfSize; // specifies the size in bytes of the bitmap file

short bfReserved1;// reserved; must be 0

short bfReserved2;// reserved; must be 0

int bfOffBits; // species the offset in bytes to the bitmap bits

}BITMAPFILEHEADER;

typedef struct tagBITMAPINFOHEADER

{

int biSize; // specifies the number of bytes required by the struct

int biWidth; // specifies width in pixels

int biHeight; // species height in pixels

short biPlanes; // specifies the number of color planes, must be 1

short biBitCount; // specifies the number of bit per pixel

int biCompression; // specifies the type of compression

int biSizeImage; // size of image in bytes

int biXPelsPerMeter;// number of pixels per meter in x axis

int biYPelsPerMeter;// number of pixels per meter in y axis

int biClrUsed; // number of colors used by the bitmap

int biClrImportant; // number of colors that are important

}BITMAPINFOHEADER;

#pragma pack(0) // restore normal structure packing rules

unsigned char\* load\_BMP\_file(char\* filename, BITMAPFILEHEADER\* bitmapFileHeader, BITMAPINFOHEADER\* bitmapInfoHeader)

{

// Open filename in read binary mode

FILE\* fp = fopen(filename,"rb");

if (fp == NULL)

{

return NULL;

}

// Read the bitmap file header

fread(bitmapFileHeader, sizeof(BITMAPFILEHEADER), 1, fp);

// Verify that this is a bmp file by checking bitmap id

if (bitmapFileHeader->bfType != 0x4D42)

{

fclose(fp);

return NULL;

}

// Read the bitmap info header

fread(bitmapInfoHeader, sizeof(BITMAPINFOHEADER), 1, fp);

// Move file pointer to the begining of bitmap data

fseek(fp, bitmapFileHeader->bfOffBits, SEEK\_SET);

// Calculate padding

int padding = 0;

while ((bitmapInfoHeader->biWidth \* 3 + padding) % 4 != 0)

{

padding++;

}

// Calculate new width, which includes padding

int width\_pad = bitmapInfoHeader->biWidth \* 3 + padding;

// Allocate memory to store image data (non-padded)

unsigned char\* bitmapImage = (unsigned char\*)malloc(bitmapInfoHeader->biWidth \* bitmapInfoHeader->biHeight \* 3 \* sizeof(unsigned char));

// Verify memory allocation

if (bitmapImage == NULL)

{

fclose(fp);

return NULL;

}

// Allocate temporary memory to read width\_pad size of data (one row)

unsigned char\* row = (unsigned char\*)malloc(width\_pad \* sizeof(unsigned int));

// Read data row by row and remove padded data

int i;

for (i = 0; i<bitmapInfoHeader->biHeight; i++) // by row

{

// Read width\_pad length of data (one row)

fread(row, sizeof(unsigned char), width\_pad, fp);

// Move pixels to bitmapImage and swap RB component (BMP stores in BGR format)

int j;

for (j = 0; j < bitmapInfoHeader->biWidth \* 3; j += 3) // by column

{

int index = (i \* bitmapInfoHeader->biWidth \* 3) + j;

bitmapImage[index + 2] = row[j + 0];

bitmapImage[index + 1] = row[j + 1];

bitmapImage[index + 0] = row[j + 2];

}

}

// Free temporary allocated data

free(row);

// Close file and return bitmap image data

fclose(fp);

return bitmapImage;

}

int save\_BMP\_file(char\* filename, BITMAPFILEHEADER\* bitmapFileHeader, BITMAPINFOHEADER\* bitmapInfoHeader, unsigned char\* bitmapImage)

{

// Open filename in write binary mode

FILE\* fp = fopen(filename,"wb");

if (fp == NULL)

{

return -1;

}

// Write the bitmap file header

fwrite(bitmapFileHeader, sizeof(BITMAPFILEHEADER), 1, fp);

// Write the bitmap info header

fwrite(bitmapInfoHeader, sizeof(BITMAPINFOHEADER), 1, fp);

// Calculate padding

int padding = 0;

while ((bitmapInfoHeader->biWidth \* 3 + padding) % 4 != 0)

{

padding++;

}

// Calculate new width, which includes padding

int width\_pad = bitmapInfoHeader->biWidth \* 3 + padding;

// Allocate temporary memory to write width\_pad size of data (one row)

unsigned char\* row = (unsigned char\*)malloc(width\_pad \* sizeof(unsigned int));

// Write dara row by row and add padded data

int i;

for (i = 0; i<bitmapInfoHeader->biHeight; i++) // by row

{

// Move pixels from bitmapImage to row and swap RB component (BMP stores in BGR format)

int j, index=0;

for (j = 0; j < bitmapInfoHeader->biWidth \* 3; j += 3)

{

index = (i \* bitmapInfoHeader->biWidth \* 3) + j;

row[j + 0] = bitmapImage[index + 2];

row[j + 1] = bitmapImage[index + 1];

row[j + 2] = bitmapImage[index + 0];

}

// Write padding zeros at the end of row

for (j = bitmapInfoHeader->biWidth \* 3; j < width\_pad; j+= 3)

{

index = (i \* bitmapInfoHeader->biWidth \* 3) + j;

row[j + 0] = 0;

row[j + 1] = 0;

row[j + 2] = 0;

}

// Write width\_pad length of data (one row)

fwrite(row, sizeof(unsigned char), width\_pad, fp);

}

// Free temporary allocated data

free(row);

if (ferror(fp))

{

fclose(fp);

return -1;

}

// Close file

fclose(fp);

return 0;

}

void black\_BMP\_except\_square(unsigned char\* bitmapImage, BITMAPINFOHEADER\* bitmapInfoHeader, int x0, int y0, int x1, int y1)

{

int i, j;

// Scan bitmapImage by pixels and replace pixels outside of square with black ( 0 )

for (i = 0; i < bitmapInfoHeader->biHeight; i++) // by row

{

for (j = 0; j < bitmapInfoHeader->biWidth \* 3; j += 3) // by column

{

// Check if the pixel is inside the square

if ((j < (x0\*3) || j > (x1\*3)) || (i < y0 || i > y1))

{

int index = (i \* bitmapInfoHeader->biWidth \* 3) + j;

bitmapImage[index + 2] = 0;

bitmapImage[index + 1] = 0;

bitmapImage[index + 0] = 0;

}

}

}

}

int main()

{

char filename[250] = "";

int x0, y0, x1, y1;

printf("Enter filename:\n");

scanf("%s", filename);

printf("Enter x0 and y0:\n");

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

printf("Enter x1 and y1:\n");

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

// Make sure the area is not a point and the width and lenght are not negative

if (x0 >= x1 || y0 >= y1)

{

printf("The area isn't correctly defined\n");

return 0;

}

// Make sure the area is a square

if ((y1 - y0) != (x1 - x0))

{

printf("The area isn't correctly defined\n");

return 0;

}

// Allocate header structures

BITMAPINFOHEADER bitmapInfoHeader;

BITMAPFILEHEADER bitmapFileHeader;

// Load BMP file from file and fill header structures, returns pointer to image data

unsigned char\* bitmapData = load\_BMP\_file(filename, &bitmapFileHeader, &bitmapInfoHeader);

if ( bitmapData == NULL )

{

printf("Cannot open file: %s\n", filename);

return 0;

}

// Bitmap image starts from bottom-left corner, our square is defined from top-left corner, so - recalculate it

int temp = y0;

y0 = bitmapInfoHeader.biHeight-y1-1;

y1 = bitmapInfoHeader.biHeight-temp-1;

// Make sure the square is inside the image

if (y1 > bitmapInfoHeader.biHeight || x1 > bitmapInfoHeader.biWidth)

{

printf("The area isn't correctly defined\n");

return 0;

}

// Fill area outside of given square with black

black\_BMP\_except\_square(bitmapData, &bitmapInfoHeader,x0,y0,x1,y1);

// Save changed bitmap image to new file. In our case we don't have to change header structures, so pass them as is

char new\_filename[] = "new\_file.bmp";

int result = save\_BMP\_file(new\_filename, &bitmapFileHeader, &bitmapInfoHeader, bitmapData);

if ( result != 0 )

{

printf("Cannot save file: %s\n", new\_filename);

return 0;

}

return result;

}