|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

**ФАКУЛЬТЕТ** ***ИУК «Информатика и управление»***

**КАФЕДРА** \_\_***ИУК4 «Программное обеспечение ЭВМ, информационные технологии»***

**ЛАБОРАТОРНАЯ РАБОТА №5**

**«Обработка бинарных файлов»**

**ДИСЦИПЛИНА: «Типы и структуры данных»**

|  |  |  |
| --- | --- | --- |
| Выполнил: студент гр. ИУК4-42Б | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Карельский М.К. )  (Подпись) |
| Проверил: | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Пчелинцева Н.И. )  (Подпись) |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | | |
|  | - Балльная оценка:  - Оценка: | |

Калуга, 2022

**Цель:** формирование практических навыков создания алгоритмов обработки бинарных файлов.

**Задание:**

1. Познакомиться со структурой бинарного bmp-файла.
2. Изучить способы программной обработки бинарного файла.
3. Реализовать алгоритм согласно варианту

**Вариант 7**

Дан BMP-файл, содержащий рисунок. Необходимо вывести размер растрового изображения в байтах, ширину и высоту изображения в пикселях.

**Листинг:**

***BMP.h***

#ifndef BMP\_H

#define BMP\_H

#include <iostream>

#include <cstdint>

#include <fstream>

#include <vector>

#include <string>

class BMP {

public:

/// Конструктор с параметрами. Загружает растровое изображение с диска

/// @param file\_path путь к файлу изображения с расширением bmp

explicit BMP(const std::string & file\_path);

/// Загрузка растрового изображения с диска

/// @param file\_path - путь для загрузки

void load(const std::string & file\_path);

/// Получение ширины растрового изображения

/// @return - текущая ширина изображения

int32\_t getWidth() const;

/// Получение высоты растрового изображения

/// @return - текущая высота изображения

int32\_t getHeight() const;

uint32\_t getSizeImage() const;

protected:

#pragma pack(push, 1)

struct FileHeader {

/// Отметка для отличия формата от других (сигнатура формата). Может содержать единственное значение 0x4D42

uint16\_t file\_type { 0x4D42 };

/// Размер файла в байтах

uint32\_t file\_size { 0 };

/// Зарезервированное поле. Всегда должно содержать ноль

uint16\_t reserved1 { 0 };

/// Зарезервированное поле. Всегда должно содержать ноль

uint16\_t reserved2 { 0 };

/// Положение пиксельных данных относительно начала данной структуры

uint32\_t offset\_data { 0 };

};

#pragma pack(pop)

struct InfoHeader {

/// Размер данной структуры в байтах, указывающий также на версию структуры

uint32\_t size { 0 };

/// Ширина растрового изображения в пикселях

int32\_t width { 0 };

/// Высота растрового изображения в пикселях

int32\_t height { 0 };

/// Количество цветовых плоскостей. Всегда должно быть равно единице

uint16\_t planes { 1 };

/// Количество бит на пиксель

uint16\_t bit\_count { 0 };

/// Способ хранения пикселей

uint32\_t compression { 0 };

/// Размер пиксельных данных в байтах

uint32\_t size\_image { 0 };

/// Количество пикселей на метр по горизонтали

int32\_t x\_pixels\_per\_meter { 0 };

/// Количество пикселей на метр по вертикали

int32\_t y\_pixels\_per\_meter { 0 };

/// Количество цветов в цветовой палитр

uint32\_t colors\_used { 0 };

/// Цвета, используемые растровым изображением

uint32\_t colors\_important { 0 };

};

struct ColorHeader {

/// Битовая маска для канала красного

uint32\_t red\_mask { 0x00ff0000 };

/// Битовая маска для канала зеленого

uint32\_t green\_mask { 0x0000ff00 };

/// Битовая маска для канала синего

uint32\_t blue\_mask { 0x000000ff };

/// Битовая маска для альфа-канала

uint32\_t alpha\_mask { 0xff000000 };

/// Тип цветового пространства. По-умолчанию sRGB (0x73524742)

uint32\_t color\_space\_type { 0x73524742 };

/// Неиспользуемые данные для цветового пространства sRGB

uint32\_t unused[16] { 0 };

};

/// Заголовочная структура текущего растрового изображения

FileHeader m\_file\_header;

/// Информационная структура текущего растрового изображения

InfoHeader m\_info\_header;

/// Цветовая структура текущего растрового изображения

ColorHeader m\_color\_header;

/// Хранилище для информации о пикселях

std::vector<uint8\_t> m\_data;

/// Количество значимых байт

uint32\_t m\_row\_stride;

/// Проверка формата цвета на соответствие форматам цветов

/// @param color\_header - ColorHeader с заданными битовыми масками и типом цветового пространства

void \_checkColorHeader(ColorHeader & color\_header);

/// Выравнивание длины строки

/// @param align\_stride - коэффициент выравнивания

/// @return - длина выравненной строки

uint32\_t \_makeStrideAligned(uint32\_t align\_stride) const;

};

#endif //BMP\_H

***BMP.cpp***

#include "BMP.h"

BMP::BMP(const std::string & file\_path) : m\_row\_stride{0} {

load(file\_path);

}

void BMP::load(const std::string & file\_path) {

// открываем файловый поток в бинарном режиме

std::ifstream input{file\_path, std::ios::binary};

// проверяем, открылся ли файл

if (!input) throw std::runtime\_error("Unable to open the input image file!");

// считываем заголовочную структуру изображения

input.read((char\*)&m\_file\_header, sizeof(m\_file\_header));

// проверяем формат изображения

if (m\_file\_header.file\_type != 0x4D42) throw std::runtime\_error("Unrecognized file format!");

// считываем информационный хэдер

input.read((char\*)&m\_info\_header, sizeof(m\_info\_header));

// если изображение 32-битное, нужно проверить и считать ColorHeader

if (m\_info\_header.bit\_count == 32) {

// проверям содержит ли файл ColorHeader (записывается только в 32-битные изображения)

if (m\_info\_header.size >= (sizeof(InfoHeader) + sizeof(ColorHeader))) {

// считываем ColorHeader

input.read((char\*)&m\_color\_header, sizeof(m\_color\_header));

// проверяем, записаны ли пиксели в формате BGRA и является ли цветовое пространство sRGB

\_checkColorHeader(m\_color\_header);

}

// если не содержит - выбрасываем исключение

else {

std::cerr << "Warning! The file \"" << file\_path << "\" does not seem to contain bit mask information\n";

throw std::runtime\_error("Error! Unrecognized file format!");

}

}

// перемещаемся к началу пикселей

input.seekg(m\_file\_header.offset\_data, std::ifstream::beg);

// в силу того, что некоторые фоторедакторы помещают в файл служебную информацию, которую можно спокойно игнорировать,

// чтобы ничего не сбилось, нужно настроить размер информационного хэдера, размер файла и указать, откуда начинаются пиксели

// если изображение 32-битное

if (m\_info\_header.bit\_count == 32) {

// записываем размер информационного хэдера, учитывая ColorHeader

m\_info\_header.size = sizeof(InfoHeader) + sizeof(ColorHeader);

// записываем, через сколько байт от начала файла начинаются пиксели, учитывая ColorHeader

m\_file\_header.offset\_data = sizeof(FileHeader) + sizeof(InfoHeader) + sizeof(ColorHeader);

}

// если изображение 24-битное

else {

// записываем размер информационного хэдера

m\_info\_header.size = sizeof(InfoHeader);

// записываем, через сколько байт от начала файла начинаются пиксели

m\_file\_header.offset\_data = sizeof(FileHeader) + sizeof(InfoHeader);

}

// записываем размер файла (это не весь размер файла, а лишь то,

// сколько занимают заголовочные структуры и служебная информация)

m\_file\_header.file\_size = m\_file\_header.offset\_data;

// если высота изображения задается отрицательным числом,

// то отсчет пикселей ведется сверху-вниз, начиная в верхнем левом углу

// если высота изображения задается положительным числом,

// то отсчет пикселей ведется снизу-вверх, начиная в нижнем левом углу

// данная программа берет за начало отсчета нижний левый угол

// проверям высоту

if (m\_info\_header.height < 0)

throw std::runtime\_error("The program can treat only BMP images with the origin in the bottom left corner!");

// выделям память для хранения информации о пикселях (ширина \* высота \* количество каналов)

m\_data.resize(m\_info\_header.width \* m\_info\_header.height \* m\_info\_header.bit\_count / 8);

// Формат изображения BMP предполагает, что каждая строка данных будет выровнена по границе четырех байтов или

// дополнена нулями, если это не так. Для изображения с разрешением 32 бита на пиксель условие выравнивания всегда выполняется.

// В случае изображений с разрешением 24 бита на пиксель условие выравнивания выполняется только в том случае,

// если ширина изображения делится на 4, в противном случае нам нужно будет заполнить строки нулями.

// если ширина изображения делится на 4

if (m\_info\_header.width % 4 == 0) {

// считываем информацию о пикселях

input.read((char\*)m\_data.data(), m\_data.size());

// обновляем размер файла (добавляем к нему то, сколько занимает информация о пикселях)

m\_file\_header.file\_size += m\_data.size();

}

// если ширина изображения не делится на 4

else {

// вычисляем width

// (количество пикселей в строке \* количество каналов = количество байт в строке (width))

m\_row\_stride = m\_info\_header.width \* m\_info\_header.bit\_count / 8;

// вычисляем line\_stride

uint32\_t new\_stride{\_makeStrideAligned(4)};

// выделяем память для отступа (line\_padding)

std::vector<uint8\_t> padding\_row(new\_stride - m\_row\_stride);

// проходимся по каждой строке пикселей

for (int y = 0; y < m\_info\_header.height; ++y) {

// считываем информацию об отступе в информацию о пикселях

// m\_data.data() - начало отсчета

// m\_row\_stride - сколько нужно отсупить, чтобы попасть на начало отступа

// y - поскольку в памяти компьютера информация хранится в виде последовательности байт (как одномерный массив),

// смена строки пикселей производится через умножение

// изначально имеем пустой вектор m\_data

// сперва считываем в него информацию о имеющихся пикселях

input.read((char\*)(m\_data.data() + m\_row\_stride \* y), m\_row\_stride);

// есть два варианта, как поступить с отступом:

// 1) считать незначащие нули

// 2) программно переместить указатель на следующую строку

// потом считываем отступ

input.read((char\*)padding\_row.data(), padding\_row.size());

}

// обновляем размер файла (размер пикселей + высота изображения \* размер отступа)

m\_file\_header.file\_size += m\_data.size() + m\_info\_header.height \* padding\_row.size();

m\_info\_header.size\_image = m\_info\_header.width \* m\_info\_header.height \* m\_info\_header.bit\_count / 8;

}

}

void BMP::\_checkColorHeader(ColorHeader &color\_header) {

ColorHeader expected\_color\_header{};

// проверяем формат цвета, чтобы он соответствовал BGRA формату

if (expected\_color\_header.red\_mask != color\_header.red\_mask ||

expected\_color\_header.blue\_mask != color\_header.blue\_mask ||

expected\_color\_header.green\_mask != color\_header.green\_mask ||

expected\_color\_header.alpha\_mask != color\_header.alpha\_mask) {

throw std::runtime\_error("Unexpected color mask format! "

"The program expects the pixel data to be in the BGRA format!");

}

// проверяем цветовое пространство, чтобы оно соответствовало sRGB

if (expected\_color\_header.color\_space\_type != color\_header.color\_space\_type) {

throw std::runtime\_error("Unexpected color space type! The program expects sRGB values");

}

}

uint32\_t BMP::\_makeStrideAligned(uint32\_t align\_stride) const {

// берем текущее количество байт в строке

uint32\_t new\_stride{m\_row\_stride};

// увеличиваем до тех пор, пока не будет делиться на 4

while (new\_stride % align\_stride != 0) ++new\_stride;

return new\_stride;

}

int32\_t BMP::getWidth() const {

return m\_info\_header.width;

}

int32\_t BMP::getHeight() const {

return m\_info\_header.height;

}

uint32\_t BMP::getSizeImage() const {

return m\_info\_header.size\_image;

}

***Menu.h***

#pragma once

#include "BMP.h"

namespace LW\_5

{

class Menu

{

public:

Menu();

Menu(std::string inputFile, std::string errorFile = "Errors.txt");

~Menu();

void Run();

private:

void Pause();

BMP\* \_bmp{};

std::string \_image{};

std::string \_errorFile{};

};

}

***Menu.cpp***

#include "Menu.h"

#include <iostream>

#include <fstream>

namespace LW\_5

{

Menu::Menu() : \_bmp(nullptr), \_image{}, \_errorFile("Errors.txt") {}

Menu::Menu(std::string inputFile, std::string errorFile) :

\_image(inputFile), \_errorFile(errorFile)

{

try

{

\_bmp = new BMP(inputFile);

}

catch (const std::exception& ex)

{

std::ofstream errorData(\_errorFile, std::ios\_base::app);

errorData << ex.what() << "\n";

errorData.close();

std::cout << ex.what() << "\n";

Pause();

throw ex;

}

}

Menu::~Menu() { remove("pauseTemp"); }

void Menu::Run()

{

uint16\_t command = 1;

while (command != 0)

{

system("cls");

std::cout << "Image: " << \_image << "\n";

std::cout << "Height: " << \_bmp->getHeight() << " px\n";

std::cout << "Width: " << \_bmp->getWidth() << " px\n";

std::cout << "Image size: " << \_bmp->getSizeImage() << " B\n\n";

std::cout << "1. Change image\n";

std::cout << "0. Exit\n";

std::cout << ">>> ";

std::cin >> command;

std::cout << "\n";

if (command == 1)

{

std::cout << "Input image path: ";

std::string path;

std::cin.ignore();

std::getline(std::cin, path);

try

{

\_bmp->load(path);

}

catch (const std::exception& ex)

{

std::ofstream errorData(\_errorFile, std::ios\_base::app);

errorData << ex.what() << "\n";

errorData.close();

std::cout << ex.what() << "\n";

Pause();

}

}

}

}

void Menu::Pause()

{

system("pause>pauseTemp");

remove("pauseTemp");

}

}

***Main.cpp***

#include "Menu.h"

#include <iostream>

int main(int argc, char\* argv[])

{

if (argc == 1)

{

std::cout << "Too few arguments\n";

}

else if (argc == 2)

{

try

{

LW\_5::Menu menu{ argv[1] };

menu.Run();

}

catch (const std::exception&)

{

return 1;

}

}

else if (argc == 3)

{

try

{

LW\_5::Menu menu{ argv[1], argv[2] };

menu.Run();

}

catch (const std::exception&)

{

return 1;

}

}

return 0;

}

**Результат:**

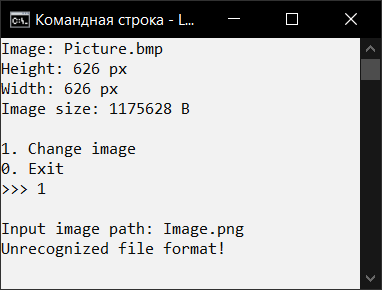


Рис. 1. Результат

**Вывод:** в ходе выполнения лабораторной работы были получены навыки загрузки и обработки BMP-файла.