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. ИУК4-31Б _____ (Суриков Н. С.)
(подпись) (Ф.И.О.)

Проверил: _____ (Былинка М. И.)
(подпись) (Ф.И.О.)

Дата сдачи (защиты): 20.12.24

Результаты сдачи (защиты):

- Балльная оценка: 95
- Оценка:

Калуга, 2024

Цель: формирование практических навыков создания алгоритмов обработки бинарных файлов.

Задачи:

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

Вариант 4

Формулировка задания:

Дан BMP-файл, содержащий рисунок. Необходимо: Изменить интенсивность цветов.

Листинг программы:

`main.cpp`

```cpp
#include "BMP/BMP.h"
#include "FileLogging/FileLogging.h"

int main(int argc, char *argv[])
{
    std::string bmp_file{};
    std::string new_bmp_file{};
    std::string error_file{"error_log.txt"};
    if (argc == 2)
    {
        bmp_file = new_bmp_file = argv[1];
    }
    else if (argc == 3)
    {
        bmp_file = argv[1];
        new_bmp_file = argv[2];
    }
    else if (argc == 4)
    {
        bmp_file = argv[1];
        new_bmp_file = argv[2];
        error_file = argv[3];
    }
    FileLogging error_log(error_file);
    try
    {
        BMP bmp(bmp_file);
        bmp.adjustColorIntensity(0.7f); // Увеличение насыщенности
```

```
29          bmp.save(new_bmp_file);
30      }
31      catch (const std::exception &e)
32      {
33          error_log.Logging(e.what());
34      }
35      return 0;
36  }
```

BMP.cpp

```
 1  #include "BMP.h"
 2  #include <algorithm>
 3  #include <fstream>
 4  #include <stdexcept>
 5
 6  // Конструктор: загрузка BMP-файла
 7  BMP::BMP(const std::string &filename)
 8  {
 9      loadFile(filename);
10  }
11
12  // Метод для изменения интенсивности цветов
13  void BMP::adjustColorIntensity(float factor)
14  {
15      for (size_t i = 0; i < pixelData.size(); i++)
16      {
17          int newValue = static_cast<int>(pixelData[i] * factor);
18          pixelData[i] = static_cast<uint8_t>(std::clamp(newValue, 0, 255));
19      }
20  }
21
22  // Метод сохранения в файл
23  void BMP::save(const std::string &outputFilename) const
24  {
25      std::ofstream outFile(outputFilename, std::ios::binary);
26      if (!outFile)
27      {
28          throw std::runtime_error("Unable to open output file");
29      }
30
31      // Заголовок файла
32      uint32_t reserved = 0;
33      uint32_t headerSize = 40;
34      uint32_t compression = 0;
35      uint32_t rowSize = ((width * 3 + 3) & ~3); // Выравнивание строки
36      uint32_t imageSize = rowSize * height;
37
38      outFile.write(reinterpret_cast<const char *>(&fileType),
sizeof(fileType));
39      outFile.write(reinterpret_cast<const char *>(&fileSize),
sizeof(fileSize));
40      outFile.write(reinterpret_cast<const char *>(&reserved),
sizeof(reserved)); // Reserved
```

```
41      outFile.write(reinterpret_cast<const char *>(&dataOffset),
sizeof(dataOffset));
42
43      // Заголовок изображения
44      outFile.write(reinterpret_cast<const char *>(&headerSize),
sizeof(headerSize));
45      outFile.write(reinterpret_cast<const char *>(&width), sizeof(width));
46      outFile.write(reinterpret_cast<const char *>(&height), sizeof(height));
47
48      uint16_t planes = 1; // Всегда 1
49      outFile.write(reinterpret_cast<const char *>(&planes), sizeof(planes));
50      outFile.write(reinterpret_cast<const char *>(&bitsPerPixel),
sizeof(bitsPerPixel));
51      outFile.write(reinterpret_cast<const char *>(&compression),
sizeof(compression));
52      outFile.write(reinterpret_cast<const char *>(&imageSize),
sizeof(imageSize));
53      uint32_t ppm = 2835; // 72 DPI в пикселях на метр
54      outFile.write(reinterpret_cast<const char *>(&ppm), sizeof(ppm));
55      outFile.write(reinterpret_cast<const char *>(&ppm), sizeof(ppm));
56      uint32_t colorsUsed = 0;
57      uint32_t importantColors = 0;
58      outFile.write(reinterpret_cast<const char *>(&colorsUsed),
sizeof(colorsUsed));
59      outFile.write(reinterpret_cast<const char *>(&importantColors),
sizeof(importantColors));
60
61      // Сохранение пикселей с учётом выравнивания
62      std::vector<uint8_t> padding(rowSize - width * 3, 0);
63      for (int y = 0; y < height; ++y)
64      {
65          size_t rowStart = y * width * 3;
66          outFile.write(reinterpret_cast<const char *>(&pixelData[rowStart]),
width * 3);
67          outFile.write(reinterpret_cast<const char *>(padding.data()),
padding.size());
68      }
69  }
70
71  // Вспомогательный метод загрузки файла
72  void BMP::loadFile(const std::string &filename)
73  {
74      std::ifstream inFile(filename, std::ios::binary);
75      if (!inFile)
76      {
77          throw std::runtime_error("Unable to open input file");
78      }
79
80      // Чтение заголовка файла
81      inFile.read(reinterpret_cast<char *>(&fileType), sizeof(fileType));
82      if (fileType != 0x4D42)
83      { // Проверка "BM" в начале файла
84          throw std::runtime_error("Unsupported file format");
85      }
86
```

```
 87      inFile.read(reinterpret_cast<char *>(&fileSize), sizeof(fileSize));
 88      inFile.ignore(4); // Пропуск резервированных байтов
 89      inFile.read(reinterpret_cast<char *>(&dataOffset), sizeof(dataOffset));
 90
 91      // Чтение заголовка изображения
 92      uint32_t headerSize;
 93      inFile.read(reinterpret_cast<char *>(&headerSize), sizeof(headerSize));
 94      if (headerSize != 40)
 95      {
 96          throw std::runtime_error("Unsupported BMP header size");
 97      }
 98
 99      inFile.read(reinterpret_cast<char *>(&width), sizeof(width));
100      inFile.read(reinterpret_cast<char *>(&height), sizeof(height));
101      inFile.ignore(2); // Пропуск количества цветовых плоскостей
102      inFile.read(reinterpret_cast<char *>(&bitsPerPixel),
sizeof(bitsPerPixel));
103      if (bitsPerPixel != 24)
104      {
105          throw std::runtime_error("Only 24-bit BMP files are supported");
106      }
107
108      inFile.ignore(24); // Пропуск оставшейся части заголовка
109      // Чтение пиксельных данных
110      uint32_t rowSize = ((width * 3 + 3) & ~3); // Выравнивание строки
111      size_t pixelDataSize = rowSize * height;
112      pixelData.resize(pixelDataSize);
113      inFile.seekg(dataOffset, std::ios::beg);
114
115      // Считываем строки с учётом padding
116      for (int y = 0; y < height; ++y)
117      {
118          inFile.read(reinterpret_cast<char *>(&pixelData[y * width * 3]),
width * 3);
119          inFile.ignore(rowSize - width * 3); // Пропуск padding
120      }
121  }
```

BMP.h

```
 1  #ifndef BMP_H
 2  #define BMP_H
 3
 4  #include <string>
 5  #include <vector>
 6  #include <cstdint>
 7
 8  class BMP {
 9  public:
10      BMP(const std::string& filename);
11      void adjustColorIntensity(float factor);
12      void save(const std::string& outputFilename) const;
13
14  private:
```

```
    // Вспомогательные методы
    void loadFile(const std::string& filename);
    void validateHeader() const;

    // Данные BMP
    uint16_t fileType;
    uint32_t fileSize;
    uint32_t dataOffset;

    uint32_t width;
    uint32_t height;
    uint16_t bitsPerPixel;

    std::vector<uint8_t> pixelData; // Пиксельные данные
};

#endif // BMP_H
```

FileLogging.cpp

```
#define _CRT_SECURE_NO_WARNINGS
#include "FileLogging.h"

FileLogging::FileLogging(std::string fileName)
{
  this->fileName = fileName;
}

void FileLogging::Logging(std::string message)
{
  std::ofstream fout(fileName, std::ios::out | std::ios::app);
  if (fout.is_open())
  {
        fout << "[" << getTime() << "] " << message << "\n";
  }
  fout.close();
}

std::string FileLogging::getTime()
{
  time_t seconds = time(nullptr);
  tm* timeinfo = localtime(&seconds);
  std::string currTime = asctime(timeinfo);
  currTime.pop_back();
  return currTime;
}
```

FileLogging.h

```
#ifndef FILE_LOGGING
#define FILE_LOGGING
#include <string>
#include <fstream>
```

```
5  #include <ctime>
6  #include <iostream>
7
8  class FileLogging
9  {
10 public:
11   FileLogging(std::string fileName);
12   void Logging(std::string message);
13
14 private:
15   std::string getTime();
16   std::string fileName;
17 };
18 #endif
```

Результат работы:

Исходник:

Результат:

Вывод: в ходе работы были сформированы практические навыки создания алгоритмов обработки бинарных файлов.