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

ОТЧЕТ

по лабораторной работе №5
по дисциплине «Алгоритмы и структуры данных»
Тема: Кодирование и декодирование

Студент гр. 9381 Колованов Р.А.

Преподаватель Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

Познакомиться с алгоритмом кодирования и декодирования Шеннона-Фано, реализовать кодирование и декодирование файлов алгоритмом Шеннона-Фано на языке C++.

Задание.

Вариант 1.

В вариантах заданий 1-ой группы (кодирование и декодирование) на вход подаётся файл с закодированным или незакодированным содержимым. Требуется раскодировать или закодировать содержимое файла определённым алгоритмом.

Кодирование: Фано-Шеннона

Описание алгоритма.

1) Алгоритм Шеннона-Фано для кодирования текста. Для этого был реализован класс *ShannonFanoEncoder*. Для начала происходит подсчет количества вхождений каждой буквы алфавита в кодируемый текст. Для этого создается вектор пар (*std::vector<std::pair<char, size_t>>*), который будет хранить буквы и количество их вхождений. Далее алгоритм проходит по каждой букве текста: если буква еще не была встречена (еще не занесена в вектор), то происходит добавление пары с этой буквой и 1 в вектор. Иначе у элемента вектора, который соответствует этой букве, инкрементируется количество вхождений. В конце вектор сортируется по убыванию значения вхождения с учетом лексикографического порядка символов алфавита. После подсчета вхождений происходит построение дерева кодирования Шеннона-Фано. Для начала в векторе частоты вхождений символов алфавита происходит поиск такого индекса k, для которого абсолютное значение разности сумм вхождений символов, стоящих слева от индекса k (включая этот индекс) и справа от индекса k, минимально. Исходный вектор вхождений делится индексом k на два вектора, которые после рекурсивно передаются в этот же метод, из которого будут возвращены созданные для этих вхождений бинарные деревья, и присваивает их

значение левому и правому поддеревьям с учетом того факта, что сумма вхождений символов у правого поддерева должна быть больше или равна сумме вхождений у левого поддерева. В процессе деления векторов вхождений символов алфавита алгоритм вскоре дойдет до ситуации, когда в векторе останется один элемент. В этом случае узлу дерева присваивается значение оставшегося символа, и происходит выход из функции.

2) Алгоритм Шеннона-Фано для декодирования текста. Для этого был реализован класс *ShannonFanoDecoder*. Для начала происходит получение бинарного дерева кодирования Шеннона-Фано и последовательности бит закодированного текста. В начале работы алгоритм находится в корне бинарного дерева Шеннона-Фано. Далее алгоритм проходит последовательность бит и при получении очередного бита выполняет следующие действия: если бит равен 0, то происходит переход в левое поддерево текущего дерева, иначе – в правое. Далее проверяется, достигнут ли лист дерева: если да – то в текущем листе записан очередной символ текста, который добавляется в раскодированный текст, после чего происходит переход обратно в корень дерева Шеннона-Фано, если же нет – то спуск по бинарному дереву происходит дальше, пока не будет встречен лист дерева.

Описание структур данных и функций.

Перечисление MessageType.

Содержит тип сообщения для логгера. В зависимости от типа сообщения меняется поток вывода, а некоторых случаях вывод может не производится. Существуют следующие значения:

- *MessageType::Common* – обычное сообщение, выводится в поток *stdout*.
- *MessageType::Warning* – предупреждающее сообщение, выводится в поток *stderr*.
- *MessageType::Error* – сообщение об ошибке, выводится в поток *stderr*.
- *MessageType::Debug* – отладочное сообщение (промежуточные данные).

Перечисление Color.

Содержит тип цвета для текста или заднего фона консоли.

Класс Logger.

Класс предоставляет функционал для вывода сообщений в консоль и файл из любой точки программы. Реализован с использованием паттерна *Singleton*. Поля и методы класса приведены в таблицах 1 и 2.

Таблица 1 - Поля класса *Logger*

Модификатор доступа	Тип и название поля	Предназначение	Значение по умолчанию
private	*int indentSize_*	Хранит размер отступа в пробелах.	*4*
private	*bool debugMode_*	Хранит информацию о том, включен ли режим отладки. При выключенном режиме отладки сообщения типа MessageType::Debug будут игнорироваться.	*false*
private	*bool fileOutput_*	Хранит информацию о том, нужно ли выводить сообщения в файл.	*false*
private	*std::ofstream file_*	Поток вывода данных в файл.	-

Таблица 2 - Методы класса *Logger*

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	*Logger&*	*getInstance()*
public	*void*	*log(const std::string& message, MessageType type = MessageType::Common, int indents = 0)*
public	*void*	*setConsoleColor(Color textColor, Color backgroundColor)*
public	*void*	*setOutputFile(const std::string& filePath)*
public	*void*	*setDebugMode(bool value)*
public	*bool*	*getDebugMode()*
public	*void*	*getCurrentDataTime()*

Метод Logger::getInstance.

Ничего не принимает. Создает статическую переменную объекта класса *Logger* (создается только один раз — при первом вызове данного метода). Возвращает ссылку на созданный объект.

Метод Logger::log.

Принимает на вход три аргумента: *message* — сообщение, *type* — тип сообщения и *indents* — количество отступов. Печатает сообщение с отступом в консоль и, если установлен флаг fileOutput_, в файл. В зависимости от типа сообщения выбирается поток вывода. Если включен режим отладки и тип сообщения — MessageType::*Debug*, то сообщение выведено не будет. Ничего не возвращает.

Метод Logger::setConsoleColor.

Принимает на вход два аргумента: *textColor* — цвет текста консоли и *backgroundColor* — цвет заднего фона текста. Меняет цвета текста и заднего фона текста консоли. Ничего не возвращает.

Метод Logger::setOutputFile.

Принимает на вход *filePath* — путь к файлу для записи сообщений. Открывает поток вывода сообщений в файл и присваивает полю *fileOutput_* значение *true*. Ничего не возвращает.

Метод Logger::setDebugMode.

Принимает на вход *value* — новое значение флага режима отладки. Устанавливает полю *silentMode_* значение *value*. Ничего не возвращает.

Метод Logger::getDebugMode.

Ничего не принимает. Возвращает значение поля *silentMode_*. Ничего не возвращает.

Метод Logger::getCurrentDataTime.

Ничего не принимает. Возвращает текущие дату и время в виде следующей строки: *<день>-<месяц>-<год>_<часы>-<минуты>-<секунды>*. Используется для генерации имени файла с логами.

Класс BinaryTree.

Класс бинарного дерева. Для реализации класса используется шаблон, который определяет тип элементов дерева. Предоставляет интерфейс для создания бинарного дерева по скобочной записи и работы с бинарным деревом. Связь элементов бинарного дерева реализована при помощи указателей. Используется для хранения дерева Шеннона-Фано. Поля и методы класса приведены в таблице 3 и 4.

Таблица 3 - Поля класса *BinaryTree*

Модификатор доступа	Тип и название поля	Предназначение	Значение по умолчанию
private	*T element*	Хранит значение корня дерева.	-
private	*BinaryTree* right*	Хранит указатель на правое поддерево.	*nullptr*
private	*BinaryTree* left*	Хранит указатель на левое поддерево.	*nullptr*

Таблица 4 - Методы класса *BinaryTree*

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	-	*BinaryTree()*
public	-	*BinaryTree(const std::string& expression)*
public	*bool*	*createFromString(const char*& character)*
public	*void*	*setElement(const T& value)*
public	*T*	*getElement()*
public	*BinaryTree**	*getRightSubtree()*
public	*BinaryTree**	*getLeftSubtree()*
public	*void*	*setRightSubtree(BinaryTree* subtree)*
public	*void*	*setLeftSubtree(BinaryTree* subtree)*
public	*bool*	*isLeaf()*
public	*std::string*	*getString() const*
public	-	*~ BinaryTree()*

Метод BinaryTree::BinaryTree.

Конструктор. Ничего не принимает. Создает пустое бинарное дерево.

Метод BinaryTree::BinaryTree.

Конструктор. Принимает на вход *expression* – строку, содержащую скобочную запись бинарного дерева. Создает бинарное дерево по скобочной записи при помощи метода *createFromString*.

Метод BinaryTree::createFromString.

Является рекурсивным методом. Принимает на вход *character* – ссылку на указатель начала строки, содержащую скобочную запись бинарного дерева. Создает бинарное дерево по заданной скобочной записи. Если бинарное дерево было успешно создано по скобочной записи, то возвращает *true*. Если скобочная запись оказалась некорректной, то возвращает *false*.

Метод BinaryTree::setElement.

Устанавливает элементу узла дерева новое значение. Принимает на вход *value* – новое значение. Ничего не возвращает.

Метод BinaryTree::getElement.

Ничего не принимает. Возвращает значение элемента узла дерева.

Метод BinaryTree::getRightSubtree.

Ничего не принимает. Возвращает правое поддерево узла дерева.

Метод BinaryTree::getLeftSubtree.

Ничего не принимает. Возвращает левое поддерево узла дерева.

Метод BinaryTree::setRightSubtree.

Устанавливает узлу дерева правое поддерево. Принимает на вход *subtree* – новое правое поддерево. Ничего не возвращает.

Метод BinaryTree::setLeftSubtree.

Устанавливает узлу дерева левое поддерево. Принимает на вход *subtree* – новое левое поддерево. Ничего не возвращает.

Метод BinaryTree::isLeaf.

Ничего не принимает. Если бинарное дерево является листом (*left = nullptr, right == nullptr*), то возвращает *true*. Иначе возвращает *false*.

Метод BinaryTree::getString.

Является рекурсивным методом. Ничего не принимает. Возвращает строку, в которой содержится скобочная запись бинарного дерева.

Метод BinaryTree::~BinaryTree.

Деструктор. Является рекурсивным методом. Очищает выделенную под элементы бинарного дерева динамическую память.

Класс Encoder.

Базовый класс кодировщика. Объявляет интерфейс, наследуемый далее конкретными кодировщиками для его реализации. Поля и методы класса приведены в таблице 5 и 6.

Таблица 5 - Поля класса *Encoder*

Модификатор доступа	Тип и название поля	Предназначение	Значение по умолчанию
protected	*CharactersFrequency frequencies_*	Хранит частоту символов в обработанном тексте.	-

Таблица 6 - Методы класса *Encoder*

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	*CharactersFrequency*	*getCharacterFrequencies()*
public	*CharactersFrequency*	*calculateTextCharacterFrequencies(const std::string& text)*
public	*BitSequence*	*encodeText(const std::string& text) = 0*

Метод Encoder:: getCharacterFrequencies.

Ничего не принимает. Возвращает частоту символов в обработанном при помощи метода *calculateTextCharacterFrequencies* тексте.

Метод Encoder:: calculateTextCharacterFrequencies.

Принимает на вход *text* – некоторый текст. Считает количество вхождений символов в текст (частоту встречи каждого символа текста) и возвращает его.

Метод Encoder:: encodeText.

Чистый виртуальный метод. Реализуется классами наследниками. Принимает на вход *text* – текст для кодирования. Используется для кодирования текста *text* некоторым алгоритмом. Возвращает закодированный текст – последовательность битов.

Класс ShannonFanoEncoder.

Класс кодировщика алгоритмом Шеннона-Фано. Используется для кодирования текста алгоритмом Шеннона-Фано. Является наследником класса *Encoder*. Поля и методы класса приведены в таблице 7 и 8.

Таблица 7 - Поля класса *ShannonFanoEncoder*

Модификатор доступа	Тип и название поля	Предназначение	Значение по умолчанию
private	*BinaryTree<char>* tree_*	Хранит дерево Шеннона-Фано для закодированного текста.	*nullptr*
private	*CharacterCodes codes_*	Хранит коды символов текста.	-

Таблица 8 - Методы класса *ShannonFanoEncoder*

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
private	*BinaryTree<char>**	*calculateCharactersTreeAndCodes(CharactersFrequency& frequency, CharacterCodes& codes, BitSequence& path)*
public	*const BinaryTree<char>**	*getTree()*
public	*CharacterCodes*	*getCharacterCodes()*
public	*BitSequence*	*encodeText(const std::string& text)*
public	-	*~ShannonFanoEncoder()*

Метод ShannonFanoEncoder::calculateCharactersTreeAndCodes.

Является рекурсивным методом. Принимает на вход три аргумента: *frequency* – частоты символов текста, которые нужно распределить в поддеревья текущего узла дерева, *codes* – ссылка на вектор кодов символов, который будет заполняться кодами по ходу работы метода, и *path* – путь от корня до текущего узла дерева в виде битовой последовательности, где 0 – это переход в левое поддерево, а 1 – в правое. Распределяет символы из вектора *frequency* при помощи алгоритма Шеннона-Фано по листьям создаваемого бинарного дерева. Возвращает созданное бинарное дерево.

Метод ShannonFanoEncoder::getTree.

Ничего не принимает. Возвращает дерево Шеннона-Фано для закодированного при помощи метода *encodeText* тексте.

Метод ShannonFanoEncoder::getCharacterCodes.

Ничего не принимает. Возвращает коды символов закодированного при помощи метода *encodeText* тексте.

Метод ShannonFanoEncoder::encodeText.

Реализация виртуального метода *encodeText*. Принимает на вход *text* – текст для кодирования. Используется для кодирования текста *text* алгоритмом Шеннона-Фано. Возвращает закодированный текст – последовательность битов.

Метод ShannonFanoEncoder::~ShannonFanoEncoder.

Деструктор. Очищает выделенную под бинарное дерево Шеннона-Фано динамическую память.

Класс Decoder.

Базовый класс декодировщика. Объявляет интерфейс, наследуемый далее конкретными декодировщиками для его реализации. Методы класса приведены в таблице 9.

Таблица 9 - Методы класса *Decoder*

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	*std::string*	*decodeText(BitSequence& sequence, BinaryTree<char>* tree) = 0*

Метод Decoder::decodeText.

Чистый виртуальный метод. Реализуется классами наследниками. Принимает на вход два аргумента: *sequence* – битовая последовательность закодированного текста и tree – дерево для декодирования. Используется для декодирования последовательности бит *sequence* в текст, закодированный некоторым алгоритмом. Возвращает декодированный текст.

Класс ShannonFanoDecoder.

Класс декодировщика алгоритмом Шеннона-Фано. Используется для декодирования текста, закодированным при помощи алгоритма Шеннона-Фано.

Является наследником класса *Decoder*. Поля и методы класса приведены в таблице 10 и 11.

Таблица 10 - Поля класса *ShannonFanoDecoder*

Модификатор доступа	Тип и название поля	Предназначение	Значение по умолчанию
private	*BinaryTree<char >* tree_*	Хранит дерево Шеннона-Фано для закодированного текста.	*nullptr*

Таблица 11 - Методы класса *ShannonFanoDecoder*

Модификатор доступа	Возвращаемое значение	Название метода и принимаемые аргументы
public	-	*ShannonFanoDecoder(const std::string& expression)*
public	*std::string*	*decodeText(BitSequence& sequence)*
public	-	*~ShannonFanoDecoder()*

Метод ShannonFanoDecoder::ShannonFanoDecoder.

Ничего не принимает. Возвращает дерево Шеннона-Фано для закодированного при помощи метода *encodeText* тексте.

Метод ShannonFanoDecoder::decodeText.

Реализация виртуального метода *decodeText*. Принимает на вход два аргумента: *sequence* – битовая последовательность закодированного текста и tree – дерево для декодирования. Используется для декодирования последовательности бит *sequence* в текст, закодированный алгоритмом Шеннона-Фано. Возвращает декодированный текст.

Метод ShannonFanoDecoder::~ShannonFanoDecoder.

Деструктор. Очищает выделенную под бинарное дерево Шеннона-Фано динамическую память.

Функция clearInput.

Очищает поток ввода *stdin* до первого символа перевода строки. Требуется для удаления некорректных символов, которые не были считаны с потока ввода. Например, когда программа ожидает получить число, а пользователь вводит букву, которая в итоге остается лежать в потоке ввода. Ничего не принимает; ничего не возвращает.

Функция main.

Для начала объявляются следующе переменные:

- *isLoopEnabled* — хранит информацию о том, надо ли продолжать выполнение основного цикла программы;
- *isDebugMode* — хранит информацию о режиме отладки;
- *logger* — хранит объект класса *Logger*.

Далее производится настройка русского языка для консоли. После у логгера *logger* вызывается метод *setOutputFile* для установки файла вывода сообщений, помимо этого устанавливается режим отладки.

Далее происходит вход в основной цикл программы. Для начала считывается выбранное пользователем действие (цифра от 1 до 6). Для выбранного действия выполняется соответствующее действие. В конце итерации основного цикла выводится результат работы программы, и программа переходит на следующую итерацию.

Выполнение работы.

Для решения поставленной задачи были написаны классы *ShannonFanoEncoder* и *ShannonFanoDecoder*, которые позволяют кодировать и декодировать текст алгоритмом Шеннона-Фано. Для хранения дерева Шеннона-Фано был реализован класс *BinaryTree*. Для вывода основной и промежуточной информации на экран и в файл был использован класс *Logger*. Для удаления некорректных символов, которые не были считаны с потока ввода, используется

функция *clearInput*. Помимо этого, был реализован CLI-интерфейс для удобной работы с программой.

Разработанный программный код см. в приложении А.

Результаты тестирования см. в приложении Б.

Сценарий использования.

Пользователь выбирает действие (вводит цифру от 1 до 6). В зависимости от выбранного действия выполняется:

- Кодирование текста, введенного с консоли. (1)
- Кодирование текста, введенного с файла. (2)
- Декодирование текста, введенного с файла. (3)
- Включение вывода промежуточных данных. (4)
- Выключение вывода промежуточных данных. (5)
- Выход из программы. (6)

Выводы.

Был изучен алгоритм кодирования и декодирования Шеннона-Фано, реализовано кодирование и декодирование файлов алгоритмом Шеннона-Фано на языке C++.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <fstream>
#include <sstream>
#include "Logger.h"
#include "Windows.h"
#include "ShannonFanoEncoder.h"
#include "ShannonFanoDecoder.h"

void clearInput() {
      // Удаляем из потока несчитанные символы до перевода на новую
строку, включая его
        std::cin.clear();
    while (std::cin.get() != '\n');
}

int main() {
      bool isLoopEnabled = true;
      bool isDebugMode = true;
      Logger& logger = Logger::getInstance();

      // Настройка русского языка
      SetConsoleCP(1251);
      SetConsoleOutputCP(1251);

      // Настройка файла вывода сообщений логгера и режима вывода
промежуточных данных
      logger.setOutputFile("Logs\\" + Logger::getCurrentDateTime() +
".log");
    logger.setDebugMode(true);

    while (isLoopEnabled) {
        HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);

        // Считывание выбора действия пользователя
        Logger::log("\nAvailable actions:\n\n  1) Encode text (input from
console).\n  2) Encode text (input from file).\n  3) Decode text (input from
file).\n");

        if (isDebugMode) {
            Logger::setConsoleColor(Color::Green, Color::Black);
        }
        Logger::log("  4) Enable output of intermediate data.\n");
        Logger::setConsoleColor(Color::LightGray, Color::Black);

        if (!isDebugMode) {
            Logger::setConsoleColor(Color::Green, Color::Black);
        }
        Logger::log("  5) Disable output of intermediate data.\n");
        Logger::setConsoleColor(Color::LightGray, Color::Black);

        Logger::log("  6) Exit.\n\n");
        std::cout << "Choose one of the actions: ";

        int action = -1;
        std::cin >> action;
        clearInput();
```

```
            while (action < 1 || action > 6) {
                Logger::log("Incorrect action. Select the action again: ");
                std::cin >> action;
                clearInput();
            }

            Logger::log("Choosed action: " + std::to_string(action) + "\n\n");

            // Кодирование текста
            if (action == 1 || action == 2) {
                std::stringstream text;

                // Считывание текста с консоли
                if (action == 1) {
                    std::string line;
                    std::cout << "Enter text: ";

                    std::getline(std::cin, line);
                    text << line;

                    Logger::log("Entered text: " + text.str() + "\n\n");
                }
                // Считывание текста с файла
                else {
                    std::ifstream inputFile("input_text.txt");

                    // Если файл не удалось открыть
                    if (!inputFile.is_open()) {
                        Logger::log("Cannot open file: input_text.txt\n");
                        continue;
                    }

                    std::string line;
                    while (!inputFile.eof()) {
                        std::getline(inputFile, line);
                        text << line << "\n";
                    }

                    Logger::log("Reading text from file
'input_text.txt'...\n[Text from file]\n" + text.str() + "\n");
                }

                ShannonFanoEncoder encoder;     // Декодировщик
                const BinaryTree<char>* tree;   // Дерево Шеннона-Фано
                CharacterCodes characterCodes;  // Коды символов
                BitSequence encodedText;        // Последовательность бит
закодированного текста

                Logger::log("Encoding text started...\n");

                // Кодирование текста
                encodedText = encoder.encodeText(text.str());
                tree = encoder.getTree();
                characterCodes = encoder.getCharacterCodes();

                Logger::log("Encoding text finished.\n\n");
                Logger::log("[Encoded text] ");
                for (auto bit : encodedText) {
                    Logger::log(std::to_string(bit));
                }
                Logger::log("\n\nSaving encoded text to file
'encoded_text.txt'.\n");

                // Сохранение результата в файл
```

```
                std::ofstream outputFile("encoded_text.txt");

                // Если файл не удалось открыть
                if (!outputFile.is_open()) {
                    Logger::log("Cannot open file: encoded_text.txt\n");
                    continue;
                }

                outputFile << tree->getString() << "\n\n";

                for (auto bit : encodedText) {
                    outputFile << bit;
                }

            }
            // Декодирование текста
            else if (action == 3) {
                std::ifstream inputFile("encoded_text.txt");

                // Если файл не удалось открыть
                if (!inputFile.is_open()) {
                    Logger::log("Cannot open file: encoded_text.txt\n");
                    continue;
                }

                std::string line("()");
                std::stringstream expression;

                // Считываем строки, содержащие скобочную запись дерева
Шеннона-Фано
                while (line != "") {
                    std::getline(inputFile, line);
                    expression << line << "\n";
                }
                Logger::log("Reading encoded text and coding tree from file
'encoded_text.txt'...\n[Coding tree] " + expression.str() + "\n");

                ShannonFanoDecoder decoder(expression.str());
                BitSequence encodedText;
                std::getline(inputFile, line); // Считываем строку, содержащую
последовательность бит закодированного текста
                Logger::log("[Encoded text] " + line + "\n\n");

                for (auto character : line) {
                    if (character == '1') {
                        encodedText.push_back(true);
                    } else if (character == '0') {
                        encodedText.push_back(false);
                    }
                }

                Logger::log("Decoding text started...\n");

                // Декодирование текста
                std::string decodedText = decoder.decodeText(encodedText);

                Logger::log("Decoding text finished.\n\n");
                Logger::log("[Decoded text]\n" + decodedText + "\n\n");
                Logger::log("Saving decoded text to file
'decoded_text.txt'.\n");

                // Сохранение результата в файл
                std::ofstream outputFile("decoded_text.txt");
```

```
            // Если файл не удалось открыть
            if (!outputFile.is_open()) {
                Logger::log("Cannot open file: decoded_text.txt\n");
                continue;
            }

            outputFile << decodedText;

        } else if (action == 4 || action == 5) {
            // Включение режима отладки (вывода промежуточной информации)
            if (action == 4) {
                isDebugMode = true;
                Logger::log("Intermediate data output enabled.\n");
            }
            // Отключение режима отладки (вывода промежуточной информации)
            else {
                isDebugMode = false;
                Logger::log("Intermediate data output disabled.\n");
            }
            logger.setDebugMode(isDebugMode);

        } else {
            // Выход из программы
            isLoopEnabled = false;
        }
    }

      return 0;
}
```

Название файла: Color.h

```
#ifndef COLOR_H
#define COLOR_H

enum class Color {
      Black = 0,
      Blue = 1,
      Green = 2,
      Cyan = 3,
      Red = 4,
      Magenta = 5,
      Brown = 6,
      LightGray = 7,
      DarkGray = 8,
      LightBlue = 9,
      LightGreen = 10,
      LightCyan = 11,
      LightRed = 12,
      LightMagenta = 13,
      Yellow = 14,
      White = 15
};

#endif // COLOR_H
```

Название файла: MessageType.h

```
#ifndef MESSAGE_TYPE_H
#define MESSAGE_TYPE_H
```

```
enum class MessageType {
    Common,
    Warning,
    Error,
    Debug
};

#endif // MESSAGE_TYPE_H
```

Название файла: Logger.h

```
#ifndef LOGGER_H
#define LOGGER_H

#include <fstream>
#include "MessageType.h"
#include "Color.h"


class Logger {
    int indentSize_ = 4;        // Размер отступа
    bool debugMode_ = false;    // Режим вывода отладочных сообщений
    bool fileOutput_ = false;   // Вывод сообщений в файл
    std::ofstream file_;        // Дескриптор выходного файла

    Logger() = default;
    Logger(const Logger&) = delete;
    Logger(Logger&&) = delete;
    Logger& operator=(const Logger&) = delete;
    Logger& operator=(Logger&&) = delete;
    ~Logger();

public:
    static Logger& getInstance();
    static void log(const std::string& message, MessageType type =
MessageType::Common, int indents = 0);
    static void setConsoleColor(Color textColor, Color backgroundColor);
    void setOutputFile(const std::string& filePath);
    void setDebugMode(bool value);
    bool getDebugMode();
    static std::string getCurrentDateTime();
};


#endif // LOGGER_H
```

Название файла: Logger.cpp

```
#include "Logger.h"
#include "Windows.h"
#include <iostream>
#include <ctime>

Logger::~Logger() {
    file_.close();
}

Logger& Logger::getInstance() {
    static Logger instance;
    return instance;
}
```

```
      void Logger::setDebugMode(bool value) {
          debugMode_ = value;
      }

      bool Logger::getDebugMode() {
          return debugMode_;
      }

      void Logger::setOutputFile(const std::string& filePath) {
          file_.close();
          file_.open(filePath);

          // Проверка открытия файла
          if (!file_.is_open()) {
              fileOutput_ = false;
              Logger::log("Cannot open file: " + filePath + "\n",
MessageType::Error);
              return;
          }

          fileOutput_ = true;
      }

      void Logger::log(const std::string& message, MessageType type, int
indents) {
          Logger& logger = Logger::getInstance();

          // Если включен режим отладки и сообщение - отладочное, то происходит
выход из функции
          if (type == MessageType::Debug && !logger.debugMode_) {
              return;
          }

          std::string indent(logger.indentSize_ * indents, ' '); // Отступ от
начала строки

          if (type == MessageType::Common || type == MessageType::Debug) {
              std::cout << indent << message;
          } else {
              std::cerr << indent << message;
          }

          if (logger.fileOutput_) {
              logger.file_ << indent << message;
          }
      }

      void Logger::setConsoleColor(Color textColor, Color backgroundColor) {
          HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
          SetConsoleTextAttribute(hConsole,
(WORD)((static_cast<int>(backgroundColor) << 4) | static_cast<int>(textColor)));
      }

      std::string Logger::getCurrentDateTime() {
          tm timeinfo;                                          // Структура с
информацией о времени
          time_t timestamp = time(nullptr);                     // Временная метка
          errno_t error = localtime_s(&timeinfo, &timestamp); // Получение
информации о времени
          char buffer[40];                                      // Буфер для
строки
```

```
        // Если возникла ошибка при получении информации о времени, то
возвращаем "00-00-00_00-00-00"
        if (error) {
            return "00-00-00_00-00-00";
        } else {
            strftime(buffer, sizeof(buffer), "%d-%m-%y_%H-%M-%S", &timeinfo);
        }

        return buffer;
    }
```

Название файла: BinaryTree.h

```
    #ifndef BINARY_TREE_H
    #define BINARY_TREE_H


    template <typename T>
    class BinaryTree {
    private:
        T element_;                     // Корень дерева
        BinaryTree* right_ = nullptr;  // Правое поддерево
        BinaryTree* left_ = nullptr;   // Левое поддерево

    public:
        BinaryTree();
        BinaryTree(const std::string& expression);
        bool createFromString(const char*& character);
        void setElement(const T& value);
        T getElement();
        BinaryTree* getRightSubtree();
        BinaryTree* getLeftSubtree();
        void setRightSubtree(BinaryTree* subtree);
        void setLeftSubtree(BinaryTree* subtree);
        bool isLeaf();
        std::string getString() const;
        ~BinaryTree();
    };

    template<>
    inline BinaryTree<char>::BinaryTree(): element_('\0') {}

    template<>
    inline BinaryTree<char>::BinaryTree(const std::string& expression) {
        const char* start = expression.c_str();
        createFromString(start);
    }

    template<>
    inline bool BinaryTree<char>::createFromString(const char*& character) {
        // Очищаем поддеревья (в случае, если до вызова метода дерево уже
использовалось)
        delete right_;
        delete left_;
        right_ = nullptr;
        left_ = nullptr;

        // Если скобочная запись начинается с '/', то это пустое БД
        if (*character == '/') {
            character++;
            return true;
        }
```

```
        // Если скобочная запись начинается с '(', то это непустое БД
        if (*character == '(') {
            character++;

            // Если нам встречается значение узла дерева, то записываем его в
узел
            if (*character != '(' && *character != ')' && *character != '/' &&
*character != '\0') {
                element_ = *character;
                character++;
            }

            // Если встречаем конец скобочной записи, то выходим
            if (*character == ')') {
                character++;
                return true;
            }

            // Создаем левое поддерево
            if (*character != '/') {
                left_ = new BinaryTree;
                bool correct = left_->createFromString(character);

                // Если не удалось корректно считать скобочную запись, то
выходим
                if (!correct) {
                    return false;
                }

            }
            else {
                character++;
            }

            // Если встречаем конец скобочной записи, то выходим
            if (*character == ')') {
                character++;
                return true;
            }

            // Создаем правое поддерево
            if (*character != '/') {
                right_ = new BinaryTree;
                bool correct = right_->createFromString(character);

                // Если не удалось корректно считать скобочную запись, то
выходим
                if (!correct) {
                    return false;
                }

            }
            else {
                character++;
            }

            // Если встречаем конец скобочной записи, то выходим
            if (*character == ')') {
                character++;
                return true;
            }
        }
```

```
    return false;
}

template<typename T>
inline void BinaryTree<T>::setElement(const T& value) {
    element_ = value;
}

template<typename T>
inline T BinaryTree<T>::getElement() {
    return element_;
}

template<typename T>
inline BinaryTree<T>* BinaryTree<T>::getRightSubtree() {
    return right_;
}

template<typename T>
inline BinaryTree<T>* BinaryTree<T>::getLeftSubtree() {
    return left_;
}

template<typename T>
inline void BinaryTree<T>::setRightSubtree(BinaryTree* subtree) {
    delete right_;
    right_ = subtree;
}

template<typename T>
inline void BinaryTree<T>::setLeftSubtree(BinaryTree* subtree) {
    delete left_;
    left_ = subtree;
}

template<>
inline bool BinaryTree<char>::isLeaf() {
    return right_ == nullptr && left_ == nullptr;
}

template <>
inline std::string BinaryTree<char>::getString() const {
    std::string result = "(";

    if (element_ != '\0') {
        result += std::string(1, element_);
    }

    if (left_ != nullptr) {
        result += left_->getString();
    } else {
        result += '/';
    }

    if (right_ != nullptr) {
        result += right_->getString();
    } else {
        result += '/';
    }

    return result + ")";
}

template <typename T>
```

```
inline BinaryTree<T>::~BinaryTree() {
    delete right_;
    delete left_;
}


#endif // BINARY_TREE_H
```

Название файла: Encoder.h

```
#ifndef ENCODER_H
#define ENCODER_H

#include <vector>
#include <string>
#include <map>
#include "BinaryTree.h"

typedef std::vector<bool> BitSequence;
// Последовательность бит - вектор значений типа bool (true - 1, false - 0)
typedef std::pair<char, size_t> CharacterFrequency;                    // Частота символа - пара из значения символа и его частоты (количество вхождений в текст)
typedef std::vector<CharacterFrequency> CharactersFrequency;     // Частота символов - вектор частоты символов
typedef std::map<char, BitSequence> CharacterCodes;                    // Коды символов - словарь, ключ которого соответствуют символу, а значение - коду этого символа


class Encoder {
protected:
      CharactersFrequency frequencies_;   // Частота символов в тексте

public:
      CharactersFrequency getCharacterFrequencies();
      CharactersFrequency calculateTextCharacterFrequencies(const std::string& text);
      virtual BitSequence encodeText(const std::string& text) = 0;
};


#endif // ENCODER_H
```

Название файла: Encoder.cpp

```
#include "Encoder.h"
#include "Logger.h"
#include <algorithm>

CharactersFrequency Encoder::calculateTextCharacterFrequencies(const std::string& text) {
      frequencies_.clear();
      Logger::log("\nCounting the frequency of characters in text...\n", MessageType::Debug);

      // Пробегаемся по символам текста и считаем их
      for (auto symbol : text) {
            size_t position = 0;
            bool inData = false;
```

```
                for (auto& element : frequencies_) {
                        if (element.first == symbol) {
                                inData = true;
                                break;
                        }

                        position++;
                }

                if (inData) {
                        frequencies_[position].second++;
                } else {
                        frequencies_.push_back(CharacterFrequency(symbol, 1));
                }
        }

        // Сортируем частоты по убыванию с учетом лексеграфического порядка
        std::sort(frequencies_.begin(), frequencies_.end(), [](const
CharacterFrequency& f1, const CharacterFrequency& f2) {
                if (f1.second != f2.second) {
                        return f1.second > f2.second;
                } else {
                        return f1.first < f2.first;
                }
        });

        Logger::log("Frequency of characters in text: ",
MessageType::Debug);
        for (auto& f : frequencies_) {
                Logger::log(std::string(1, f.first) + "(" +
std::to_string(f.second) + ") ", MessageType::Debug);
        }
        Logger::log("\n", MessageType::Debug);

        return frequencies_;
    }

    CharactersFrequency Encoder::getCharacterFrequencies() {
        return frequencies_;
    }
```

Название файла: Decoder.h

```
    #ifndef DECODER_H
    #define DECODER_H

    #include <vector>
    #include <string>
    #include <map>
    #include "BinaryTree.h"

    typedef std::vector<bool> BitSequence;
    typedef std::map<char, BitSequence> CharacterCodes;


    class Decoder {
    public:
        virtual std::string decodeText(BitSequence& sequence,
BinaryTree<char>* tree) = 0;
    };
```

```
#endif // DECODER_H
```

Название файла: ShannonFanoEncoder.h

```
#ifndef SHANNON_FANO_ENCODER_H
#define SHANNON_FANO_ENCODER_H

#include "Encoder.h"
#include "BinaryTree.h"


class ShannonFanoEncoder: public Encoder {
private:
	BinaryTree<char>* tree_ = nullptr;  // Дерево Шеннона-Фано
	CharacterCodes codes_;                    // Коды символов алфавита

	BinaryTree<char>*
calculateCharactersTreeAndCodes(CharactersFrequency& frequency, CharacterCodes&
codes, BitSequence& path);

public:
	const BinaryTree<char>* getTree();
	CharacterCodes getCharacterCodes();
	BitSequence encodeText(const std::string& text);
	~ShannonFanoEncoder();
};


#endif // SHANNON_FANO_ENCODER_H
```

Название файла: ShannonFanoEncoder.cpp

```
#include "ShannonFanoEncoder.h"
#include "Logger.h"
#include <iostream>
#include <sstream>
#include <cmath>

BinaryTree<char>*
ShannonFanoEncoder::calculateCharactersTreeAndCodes(CharactersFrequency&
frequency, CharacterCodes& codes, BitSequence& path) {
	BinaryTree<char>* tree = new BinaryTree<char>;
	CharactersFrequency left;
	CharactersFrequency right;
	long long minDelta = LLONG_MAX;

	std::string pathString;
	for (size_t i = 0; i < path.size(); i++) {
		pathString += std::to_string(path[i]);
	}

	// Если кодировать нечего - то возвращается пустое дерево
	if (frequency.size() == 0) {
		return tree;
	}
	// Если в списке частоты символов осталось одно значение, то
происходит присваивание текущему узлу дерева символа
	// и добавление кода текущего символа в массив кодов символов
алфавита
	else if (frequency.size() == 1) {
```

```
                if (path.size() == 0) {
                      path.push_back(false);

      tree->setLeftSubtree(calculateCharactersTreeAndCodes(frequency, codes,
path));
                      path.pop_back();
                } else {
                      Logger::log("Placing character '" + std::string(1,
frequency[0].first) + "' to node " + pathString + "\n", MessageType::Debug,
path.size());
                      codes[frequency[0].first] = path;
                      tree->setElement(frequency[0].first);
                }
                return tree;
           }
           // Если в списке частоты символов осталось более одного значение, то
происходит разделение этого списка на два
           else {
                size_t middleIndex = 0;
                long long leftSum = 0;
                long long rightSum = 0;

                for (size_t i = 0; i < frequency.size(); i++) {
                      rightSum += frequency[i].second;
                }

                // Находим такой k, при котором различие между суммой частот
двух списков минимально
                for (size_t k = 0; k < frequency.size(); k++) {
                      leftSum += frequency[k].second;
                      rightSum -= frequency[k].second;

                      if (abs(rightSum - leftSum) < abs(minDelta)) {
                            middleIndex = k;
                            minDelta = rightSum - leftSum;
                      }
                }

                // Заполняем левый и правый подсписки списка
                for (size_t i = 0; i <= middleIndex; i++) {
                      left.push_back(frequency[i]);
                }
                for (size_t i = middleIndex + 1; i < frequency.size(); i++) {
                      right.push_back(frequency[i]);
                }

                leftSum = 0;
                rightSum = 0;

                Logger::log("Characters frequency: ", MessageType::Debug,
path.size());
                for (size_t i = 0; i < left.size(); i++) {
                      Logger::log(std::string(1, left[i].first),
MessageType::Debug);
                      leftSum += left[i].second;
                }
                Logger::log("(" + std::to_string(leftSum) + ") ",
MessageType::Debug);
                for (size_t i = 0; i < right.size(); i++) {
                      Logger::log(std::string(1, right[i].first),
MessageType::Debug);
                      rightSum += right[i].second;
                }
```

```
                  Logger::log("(" + std::to_string(rightSum) + ")\n",
MessageType::Debug);
            }

            // Если правая сумма меньше левой, то меняем их местами
            if (minDelta < 0) {
                  std::swap(left, right);
            }

            // Создаем левое поддерево
            Logger::log("Creating left subtree (" + pathString + "0):\n",
MessageType::Debug, path.size());
            path.push_back(false);
            tree->setLeftSubtree(calculateCharactersTreeAndCodes(left, codes,
path));
            path.pop_back();

            // Создаем правое поддерево
            Logger::log("Creating right subtree (" + pathString + "1):\n",
MessageType::Debug, path.size());
            path.push_back(true);
            tree->setRightSubtree(calculateCharactersTreeAndCodes(right, codes,
path));
            path.pop_back();

            return tree;
      }

      const BinaryTree<char>* ShannonFanoEncoder::getTree() {
            return tree_;
      }

      CharacterCodes ShannonFanoEncoder::getCharacterCodes() {
            return codes_;
      }

      BitSequence ShannonFanoEncoder::encodeText(const std::string& text) {
            BitSequence encodedText;

            // Получаем частоту символов текста
            calculateTextCharacterFrequencies(text);

            // Строим дерево Шеннона-Фано
            Logger::log("\nBuilding Shennon-Fano tree...\n",
MessageType::Debug);
            delete tree_;
            tree_ = calculateCharactersTreeAndCodes(frequencies_, codes_,
encodedText);
            encodedText.clear();

            Logger::log("\nReplace text characters with their codes:\n",
MessageType::Debug);
            // Пробегаемся по символам текста и кодируем их
            for (auto character : text) {
                  std::stringstream codeString;
                  BitSequence& code = codes_[character];

                  for (size_t i = 0; i < code.size(); i++) {
                        encodedText.push_back(code[i]);
                        codeString << code[i];
                  }

                  Logger::log("'" + std::string(1, character) + "' -> " +
codeString.str() + "\n", MessageType::Debug);
```

```
        }
        Logger::log("\n", MessageType::Debug);

        return encodedText;
    }

    ShannonFanoEncoder::~ShannonFanoEncoder() {
        delete tree_;
    }
```

Название файла: ShannonFanoDecoder.h

```
#ifndef SHANNON_FANO_DECODER_H
#define SHANNON_FANO_DECODER_H

#include "Decoder.h"


class ShannonFanoDecoder {
private:
    BinaryTree<char>* tree_ = nullptr;  // Дерево Шеннона-Фано

public:
    ShannonFanoDecoder(const std::string& expression);
    std::string decodeText(BitSequence& sequence);
    ~ShannonFanoDecoder();
};


#endif // SHANNON_FANO_DECODER_H
```

Название файла: ShannonFanoDecoder.cpp

```
#include "ShannonFanoDecoder.h"
#include "Logger.h"
#include <sstream>

std::string ShannonFanoDecoder::decodeText(BitSequence& sequence) {
    std::stringstream characterStream;
    BinaryTree<char>* subtree = tree_;

    // Проход по битам закодированного текста
    for (auto bit : sequence) {
        Logger::log(std::to_string(bit), MessageType::Debug);

        // В зависимости от значения бита происходит переход либо в
правое поддерево, либо в левое поддерево
        if (bit) {
            subtree = subtree->getRightSubtree();
        } else {
            subtree = subtree->getLeftSubtree();
        }

        // Если достгнут лист дерева (очередной символ текста), то он
добавляется в текст, и происходит переход в корень дерева
        if (subtree->isLeaf()) {
            Logger::log(" -> '" + std::string(1,
subtree->getElement()) + "'\n", MessageType::Debug);
            characterStream << subtree->getElement();
            subtree = tree_;
        }
```

```
        }

        return characterStream.str();
}

ShannonFanoDecoder::ShannonFanoDecoder(const std::string& expression) {
        tree_ = new BinaryTree<char>(expression);
}

ShannonFanoDecoder::~ShannonFanoDecoder() {
        delete tree_;
}
```

ПРИЛОЖЕНИЕ Б
ТЕСТИРОВАНИЕ

Таблица Б.1 - Примеры тестовых случаев на некорректных данных

№ п/п	Входные данные	Выходные данные
1.	Choose one of the actions: -4	Incorrect action. Select the action again:
2.	Choose one of the actions: 7	Incorrect action. Select the action again:
3.	Choose one of the actions: 2 Choosed action: 2	Cannot open file: input_text.txt
4.	Choose one of the actions: 3 Choosed action: 3	Cannot open file: encoded_text.txt

Таблица Б.2 - Примеры тестовых случаев на корректных данных

№ п/п	Входные данные	Выходные данные
5.	Choose one of the actions: 1 Choosed action: 1 Enter text: Hello, мир! Entered text: Hello, мир!	[Encoded text] 1101110000011111011100011001001111010 Saving encoded text to file 'encoded_text.txt'.
6.	Choose one of the actions: 3 Choosed action: 3 Reading encoded text and coding tree from file 'encoded_text.txt'... [Coding tree] (((l//)((и//)((м//)(р//))))(((//)((!//)(,//)))((Н//)((е//)(о//))))) [Encoded text] 1101110000011111011100011001001111010	[Decoded text] Hello, мир! Saving decoded text to file 'decoded_text.txt'.
7.	Choose one of the actions: 1 Choosed action: 1 Enter text: Entered text:	[Encoded text] Saving encoded text to file 'encoded_text.txt'.
8.	Choose one of the actions: 1 Choosed action: 1 Enter text: ? Entered text: ?	[Encoded text] 0 Saving encoded text to file 'encoded_text.txt'.

Таблица Б.2 - Примеры тестовых случаев на корректных данных

№ п/п	Входные данные	Выходные данные
9.	Choose one of the actions: 1 Choosed action: 1 Enter text: Аб Entered text: Аб	[Encoded text] 01 Saving encoded text to file 'encoded_text.txt'.
10.	Choose one of the actions: 2 Choosed action: 2 Reading text from file 'input_text.txt'... [Text from file] Текст с файла!	[Encoded text] 000100010100111100001011001110101010011101011011111110 Saving encoded text to file 'encoded_text.txt'.
11.	Choose one of the actions: 3 Choosed action: 3 Reading encoded text and coding tree from file 'encoded_text.txt'... [Coding tree] ((((T//)(//))((a//)(c//)))((((e//)(й//))((к//)(л//)))(((т//)(ф//))((//)(!//))))) [Encoded text] 000100010100111100001011001110101010011101011011111110	[Decoded text] Текст с файла! Saving decoded text to file 'decoded_text.txt'.