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

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Алгоритмы и структуры данных»
Тема: Алгоритмы сортировки

Студент гр. 9381 _____ Прибылов Н.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

Изучить различные алгоритмы сортировки.

Задание.

Вариант 5

Гибрид сортировки пузырьком и сортировки выбором; шейкерная сортировка.

Основные теоретические положения.

Сортировка пузырьком (Bubble sort) — простейшая сортировка. Работает следующим образом: будем идти по массиву слева направо. Если текущий элемент больше следующего, меняем их местами. Делаем так, пока массив не будет отсортирован. Заметим, что после первой итерации самый большой элемент будет находиться в конце массива, на правильном месте. После двух итераций на правильном месте будут стоять два наибольших элемента, и так далее. Очевидно, не более чем после n итераций массив будет отсортирован. Таким образом, асимптотика в худшем и среднем случае – $O(n^2)$, в лучшем случае – $O(n)$.

Достоинства — простая реализация. Недостатки — крайне неэффективная.

Сортировка выбором (Selection sort) заключается в следующем: на очередной итерации будем находить минимум в массиве после текущего элемента и менять его с ним, если надо. Таким образом, после i-ой итерации первые i элементов будут стоять на своих местах. Асимптотика: $O(n^2)$ в лучшем, среднем и худшем случае.

Достоинства — эффективнее и быстрее, чем пузырьковая. Недостатки — неустойчива (хотя есть и устойчивые реализации).

Шейкерная сортировка (Cocktail shaker sort), также известная как сортировка перемешиванием и коктейльная сортировка. Является улучшением пузырьковой: после прохода массива слева направо, не возвращаемся сразу в начало, а идём в обратную сторону — справа налево. Асимптотика у алгоритма такая же, как и у сортировки пузырьком, однако реальное время работы лучше.

Достоинства — работает быстрее пузырьковой. Недостатки — по сравнению с другими сортировками, всё равно неэффективна.

Описание алгоритмов.

Гибрид сортировки пузырьком и сортировки выбором:

Часть массива сортируется сортировкой выбором: На очередной итерации будет находиться минимум в массиве после текущего элемента и менять его с ним, если надо. В итоге, после i-ой итерации первые i элементов будут стоять на своих местах.

Оставшаяся часть сортируется пузырьком: проходит до конца второй части, меняя местами неотсортированные соседние элементы друг с другом. В результате первого прохода на последнем месте окажется максимальный элемент. Далее снова обходит неотсортированную часть массива (от первого элемента до предпоследнего) и меняет по пути неотсортированных соседей. Второй по величине элемент окажется на предпоследнем месте. И так, будет обходить всё уменьшающуюся неотсортированную часть массива, перемещая найденные максимумы в конец.

Шейкерная сортировка:

Как и в пузырьковой, алгоритм проходит по массиву, меняя, если нужно, соседние элементы друг с другом (таким образом, после прохода, максимальный элемент оказывается в конце). Но, оказавшись на конце массива, не начинает проход с начала, как в пузырьковой, а разворачивается и проходит в обратную сторону (соответственно, после прохода, минимальный элемент

оказывается в начале). После каждого прохода в одну сторону, запоминает место, где последний раз произошла замена, т. к. если при движении по части массива перестановки не происходят, то эта часть массива уже отсортирована и, следовательно, её можно исключить из рассмотрения.

Описание структур данных и функций.

void printTask(); // печатает задание

void printMenu(); // печатает меню

void printVector(const std::vector<int>& vec); // принимает вектор и печатает его

void shakerSort(std::vector<int>& vec); // принимает вектор и применяет шейкерную сортировку

void bubbleSelectionSort(std::vector<int>& vec); // принимает вектор и применяет сортировку пузырьком/выбором

void menu(); // вызывает меню

void consoleInput(); // организует ввод с консоли

void fileInput(); // организует ввод с файла

void performTask(std::istream& infile); // принимает поток чтения, начинает работу программы

class Logger — вспомогательный класс для логгирования промежуточных результатов.

Методы класса:

static Logger& instance() — возвращает экземпляр класса.

void log(const std::string& str, bool toConsole = true, bool toFile = true) — принимает строку, которую нужно внести в лог, и две опции — печатать в консоль и/или в файл.

Logger() — конструктор, создаёт файл лога и открывает его.

~Logger()— деструктор, закрывает файл лога.

Конструкторы копирования, перемещения, операторы присваивания объявлены удалёнными во избежание случайного дублирования экземпляра класса.

Разработанный программный код см. в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные
1	23 69 45	Введён вектор: 23 69 45 Сортировка пузырьком/выбором: Вектор: 23 45 69 Шейкерная сортировка: Вектор: 23 45 69 std::sort: Вектор: 23 45 69
2	-10 20 -30 40 -50 60 -70	Введён вектор: -10 20 -30 40 -50 60 -70 Сортировка пузырьком/выбором: Вектор: -70 20 -30 40 -50 60 -10 Вектор: -70 -50 -30 40 20 60 -10 Вектор: -70 -50 -30 -10 20 60 40 Вектор: -70 -50 -30 -10 20 40 60 Шейкерная сортировка: Вектор: -10 -30 20 40 -50 60 -70 Вектор: -10 -30 20 -50 40 60 -70 Вектор: -10 -30 20 -50 40 -70 60 Вектор: -10 -30 20 -50 -70 40 60 Вектор: -10 -30 20 -70 -50 40 60 Вектор: -10 -30 -70 20 -50 40 60

		Вектор: -10 -70 -30 20 -50 40 60 Вектор: -70 -10 -30 20 -50 40 60 Вектор: -70 -30 -10 20 -50 40 60 Вектор: -70 -30 -10 -50 20 40 60 Вектор: -70 -30 -50 -10 20 40 60 Вектор: -70 -50 -30 -10 20 40 60 std::sort: Вектор: -70 -50 -30 -10 20 40 60
3	5 6 -8 -7 7 8 -6 -5 1 2 -4 -3 3 4 -2 -1	Введён вектор: 5 6 -8 -7 7 8 -6 -5 1 2 -4 -3 3 4 -2 -1 Сортировка пузырьком/выбором: Вектор: -8 6 5 -7 7 8 -6 -5 1 2 -4 -3 3 4 -2 -1 Вектор: -8 -7 5 6 7 8 -6 -5 1 2 -4 -3 3 4 -2 -1 Вектор: -8 -7 -6 6 7 8 5 -5 1 2 -4 -3 3 4 -2 -1 Вектор: -8 -7 -6 -5 7 8 5 6 1 2 -4 -3 3 4 -2 -1 Вектор: -8 -7 -6 -5 -4 8 5 6 1 2 7 -3 3 4 -2 -1 Вектор: -8 -7 -6 -5 -4 -3 5 6 1 2 7 8 3 4 -2 -1 Вектор: -8 -7 -6 -5 -4 -3 -2 6 1 2 7 8 3 4 5 -1 Вектор: -8 -7 -6 -5 -4 -3 -2 -1 1 2 7 8 3 4 5 6 Вектор: -8 -7 -6 -5 -4 -3 -2 -1 1 2 3 8 7 4 5 6 Вектор: -8 -7 -6 -5 -4 -3 -2 -1 1 2 3 7 8 4 5 6 Вектор: -8 -7 -6 -5 -4 -3 -2 -1 1 2 3 7 4 8 5 6 Вектор: -8 -7 -6 -5 -4 -3 -2 -1 1 2 3 7 4 5 8 6 Вектор: -8 -7 -6 -5 -4 -3 -2 -1 1 2 3 7 4 5 6 8 Вектор: -8 -7 -6 -5 -4 -3 -2 -1 1 2 3 4 7 5 6 8 Вектор: -8 -7 -6 -5 -4 -3 -2 -1 1 2 3 4 5 7 6 8 Вектор: -8 -7 -6 -5 -4 -3 -2 -1 1 2 3 4 5 6 7 8 Шейкерная сортировка: Вектор: 5 -8 -7 6 7 8 -6 -5 1 2 -4 -3 3 4 -2 -1 Вектор: 5 -8 -7 6 7 -6 8 -5 1 2 -4 -3 3 4 -2 -1 Вектор: 5 -8 -7 6 7 -6 -5 8 1 2 -4 -3 3 4 -2 -1 Вектор: 5 -8 -7 6 7 -6 -5 1 8 2 -4 -3 3 4 -2 -1 Вектор: 5 -8 -7 6 7 -6 -5 1 2 8 -4 -3 3 4 -2 -1 Вектор: 5 -8 -7 6 7 -6 -5 1 2 -4 8 -3 3 4 -2 -1 Вектор: 5 -8 -7 6 7 -6 -5 1 2 -4 -3 8 3 4 -2 -1

		Вектор: 5 -8 -7 6 7 -6 -5 1 2 -4 -3 3 8 4 -2 -1 Вектор: 5 -8 -7 6 7 -6 -5 1 2 -4 -3 3 4 8 -2 -1 Вектор: 5 -8 -7 6 7 -6 -5 1 2 -4 -3 3 4 -2 8 -1 Вектор: 5 -8 -7 6 7 -6 -5 1 2 -4 -3 3 4 -2 -1 8 Вектор: 5 -8 -7 6 7 -6 -5 1 2 -4 -3 3 -2 4 -1 8 Вектор: 5 -8 -7 6 7 -6 -5 1 2 -4 -3 -2 3 4 -1 8 Вектор: 5 -8 -7 6 7 -6 -5 1 -4 2 -3 -2 3 4 -1 8 Вектор: 5 -8 -7 6 7 -6 -5 -4 1 2 -3 -2 3 4 -1 8 Вектор: 5 -8 -7 6 -6 7 -5 -4 1 2 -3 -2 3 4 -1 8 Вектор: 5 -8 -7 -6 6 7 -5 -4 1 2 -3 -2 3 4 -1 8 Вектор: -8 5 -7 -6 6 7 -5 -4 1 2 -3 -2 3 4 -1 8 Вектор: -8 -7 5 -6 6 7 -5 -4 1 2 -3 -2 3 4 -1 8 Вектор: -8 -7 -6 5 6 7 -5 -4 1 2 -3 -2 3 4 -1 8 Вектор: -8 -7 -6 5 6 -5 7 -4 1 2 -3 -2 3 4 -1 8 Вектор: -8 -7 -6 5 6 -5 -4 7 1 2 -3 -2 3 4 -1 8 Вектор: -8 -7 -6 5 6 -5 -4 1 7 2 -3 -2 3 4 -1 8 Вектор: -8 -7 -6 5 6 -5 -4 1 2 7 -3 -2 3 4 -1 8 Вектор: -8 -7 -6 5 6 -5 -4 1 2 -3 7 -2 3 4 -1 8 Вектор: -8 -7 -6 5 6 -5 -4 1 2 -3 -2 7 3 4 -1 8 Вектор: -8 -7 -6 5 6 -5 -4 1 2 -3 -2 3 7 4 -1 8 Вектор: -8 -7 -6 5 6 -5 -4 1 2 -3 -2 3 4 7 -1 8 Вектор: -8 -7 -6 5 6 -5 -4 1 2 -3 -2 3 4 -1 7 8 Вектор: -8 -7 -6 5 6 -5 -4 1 2 -3 -2 3 -1 4 7 8 Вектор: -8 -7 -6 5 6 -5 -4 1 2 -3 -2 -1 3 4 7 8 Вектор: -8 -7 -6 5 6 -5 -4 1 -3 2 -2 -1 3 4 7 8 Вектор: -8 -7 -6 5 6 -5 -4 -3 1 2 -2 -1 3 4 7 8 Вектор: -8 -7 -6 5 -5 6 -4 -3 1 2 -2 -1 3 4 7 8 Вектор: -8 -7 -6 -5 5 6 -4 -3 1 2 -2 -1 3 4 7 8 Вектор: -8 -7 -6 -5 5 -4 6 -3 1 2 -2 -1 3 4 7 8 Вектор: -8 -7 -6 -5 5 -4 -3 6 1 2 -2 -1 3 4 7 8 Вектор: -8 -7 -6 -5 5 -4 -3 1 6 2 -2 -1 3 4 7 8 Вектор: -8 -7 -6 -5 5 -4 -3 1 2 6 -2 -1 3 4 7 8 Вектор: -8 -7 -6 -5 5 -4 -3 1 2 -2 6 -1 3 4 7 8 Вектор: -8 -7 -6 -5 5 -4 -3 1 2 -2 -1 6 3 4 7 8

		Вектор: -8 -7 -6 -5 5 -4 -3 1 2 -2 -1 3 6 4 7 8 Вектор: -8 -7 -6 -5 5 -4 -3 1 2 -2 -1 3 4 6 7 8 Вектор: -8 -7 -6 -5 5 -4 -3 1 -2 2 -1 3 4 6 7 8 Вектор: -8 -7 -6 -5 5 -4 -3 -2 1 2 -1 3 4 6 7 8 Вектор: -8 -7 -6 -5 -4 5 -3 -2 1 2 -1 3 4 6 7 8 Вектор: -8 -7 -6 -5 -4 -3 5 -2 1 2 -1 3 4 6 7 8 Вектор: -8 -7 -6 -5 -4 -3 -2 5 1 2 -1 3 4 6 7 8 Вектор: -8 -7 -6 -5 -4 -3 -2 1 5 2 -1 3 4 6 7 8 Вектор: -8 -7 -6 -5 -4 -3 -2 1 2 5 -1 3 4 6 7 8 Вектор: -8 -7 -6 -5 -4 -3 -2 1 2 -1 5 3 4 6 7 8 Вектор: -8 -7 -6 -5 -4 -3 -2 1 2 -1 3 5 4 6 7 8 Вектор: -8 -7 -6 -5 -4 -3 -2 1 2 -1 3 4 5 6 7 8 Вектор: -8 -7 -6 -5 -4 -3 -2 1 -1 2 3 4 5 6 7 8 Вектор: -8 -7 -6 -5 -4 -3 -2 -1 1 2 3 4 5 6 7 8 std::sort: Вектор: -8 -7 -6 -5 -4 -3 -2 -1 1 2 3 4 5 6 7 8
4	9 8 7 6 5 4 3 2 1	Введён вектор: 9 8 7 6 5 4 3 2 1 Сортировка пузырьком/выбором: Вектор: 1 8 7 6 5 4 3 2 9 Вектор: 1 2 7 6 5 4 3 8 9 Вектор: 1 2 3 6 5 4 7 8 9 Вектор: 1 2 3 4 5 6 7 8 9 Шейкерная сортировка: Вектор: 8 9 7 6 5 4 3 2 1 Вектор: 8 7 9 6 5 4 3 2 1 Вектор: 8 7 6 9 5 4 3 2 1 Вектор: 8 7 6 5 9 4 3 2 1 Вектор: 8 7 6 5 4 9 3 2 1 Вектор: 8 7 6 5 4 3 9 2 1 Вектор: 8 7 6 5 4 3 2 9 1 Вектор: 8 7 6 5 4 3 2 1 9 Вектор: 8 7 6 5 4 3 1 2 9 Вектор: 8 7 6 5 4 1 3 2 9 Вектор: 8 7 6 5 1 4 3 2 9

		Вектор: 8 7 6 1 5 4 3 2 9 Вектор: 8 7 1 6 5 4 3 2 9 Вектор: 8 1 7 6 5 4 3 2 9 Вектор: 1 8 7 6 5 4 3 2 9 Вектор: 1 7 8 6 5 4 3 2 9 Вектор: 1 7 6 8 5 4 3 2 9 Вектор: 1 7 6 5 8 4 3 2 9 Вектор: 1 7 6 5 4 8 3 2 9 Вектор: 1 7 6 5 4 3 8 2 9 Вектор: 1 7 6 5 4 3 2 8 9 Вектор: 1 7 6 5 4 2 3 8 9 Вектор: 1 7 6 5 2 4 3 8 9 Вектор: 1 7 6 2 5 4 3 8 9 Вектор: 1 7 2 6 5 4 3 8 9 Вектор: 1 2 7 6 5 4 3 8 9 Вектор: 1 2 6 7 5 4 3 8 9 Вектор: 1 2 6 5 7 4 3 8 9 Вектор: 1 2 6 5 4 7 3 8 9 Вектор: 1 2 6 5 4 3 7 8 9 Вектор: 1 2 6 5 3 4 7 8 9 Вектор: 1 2 6 3 5 4 7 8 9 Вектор: 1 2 3 6 5 4 7 8 9 Вектор: 1 2 3 5 6 4 7 8 9 Вектор: 1 2 3 5 4 6 7 8 9 Вектор: 1 2 3 4 5 6 7 8 9 std::sort: Вектор: 1 2 3 4 5 6 7 8 9
5	1 2 3 4 5 6 7 8 9	Введён вектор: 1 2 3 4 5 6 7 8 9 Сортировка пузырьком/выбором: Вектор: 1 2 3 4 5 6 7 8 9 Шейкерная сортировка: Вектор: 1 2 3 4 5 6 7 8 9 std::sort: Вектор: 1 2 3 4 5 6 7 8 9

6	90 -90	Введён Вектор: 90 -90 Сортировка пузырьком/выбором: Вектор: -90 90 Шейкерная сортировка: Вектор: -90 90 std::sort: Вектор: -90 90

Выводы.

Были изучены и реализованы алгоритмы сортировки — пузырьковая/вставками и шейкерная.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <algorithm>
#include "Logger.h"

char kDefaultStopString[] = "STOP";
char kDefaultFileName[] = "input.txt";

void printTask(); // печатает задание
void printMenu(); // печатает меню
void printVector(const std::vector<int>& vec); // печатает вектор
void shakerSort(std::vector<int>& vec); // шейкерная сортировка
void  bubbleSelectionSort(std::vector<int>&  vec);  //  сортировка пузырьком/выбором
void menu(); // вызывает меню
void consoleInput(); // организует ввод с консоли
void fileInput(); // организует ввод с файла
void performTask(std::istream& infile); // принимает поток чтения, начинает работу программы


void printTask() {
    Logger::instance().log("Сортировка пузырьком/выбором, шейкерная сортировка.\n");
}

void printMenu() {
    std::cout << "1. Ввести данные с клавиатуры.\n"
                 "2. Ввести данные с файла.\n"
                 "0. Выход из программы.\n";
}

void printVector(const std::vector<int>& vec) {
    Logger::instance().log("Вектор: ");
    for (auto v : vec) {
        Logger::instance().log(std::to_string(v) + " ");
    }
    Logger::instance().log("\n");
}

void shakerSort(std::vector<int>& vec) {
    if (vec.size() < 2) return;
    size_t left = 0;
    size_t right = vec.size() - 1;
    size_t control = vec.size() - 1;
    while (left < right) {
        // поиск минимального элемента с начала
        for (size_t i = left; i < right; i++) {
            if (vec[i] > vec[i+1]) {
```

```
                            // меняются два соседних элемента, как в
пузырьковой сортировке
                        std::swap(vec[i], vec[i+1]);
                        printVector(vec);
                        control = i;
                    }
                }
                 right = control; // контроль отсортированной части (чтобы
не пробегать по ней лишний раз)
                // поиск максимального элемента с конца
                for (size_t i = right; i > left; i--) {
                    if (vec[i] < vec[i-1]) {
                        //
                        std::swap(vec[i], vec[i-1]);
                        printVector(vec);
                        control = i;
                    }
                }
                left = control; // контроль отсортированной части (чтобы не
пробегать по ней лишний раз)
            }
        }

        void bubbleSelectionSort(std::vector<int>& vec) {
            if (vec.size() < 2) return;
            size_t begin = 0;
            size_t end = vec.size();
            size_t mid = (end-1)*3/4;
            // часть вектора сортируется сортировкой выбором
            for (size_t i = begin; i < mid; i++) {
                int jmin = i;
                for (size_t j = i+1; j < end; j++) {
                    if (vec.at(j) < vec.at(jmin)) {
                        jmin = j;
                    }
                }
                if (jmin != i) {
                    // минимальный элемент вставляется в начало вектора
                    std::swap(vec[i], vec[jmin]);
                    printVector(vec);
                }
            }

            // оставшаяся часть сортируется пузырьковой сортировкой
            for (size_t i = mid; i < end-1; i++) {
                for (size_t j = mid; j < end-i+mid-1; j++) {
                    if (vec.at(j) > vec.at(j+1)) {
                        // меняются два соседних элемента
                        std::swap(vec[j], vec[j+1]);
                        printVector(vec);
                    }
                }
            }
        }

        void menu() {
            printTask();
            printMenu();
```

```
        char c = '1';
        do {
            std::cin >> c;
            std::cin.ignore(256, '\n');
            switch(c) {
                case '1':
                    consoleInput();
                    break;
                case '2':
                    fileInput();
                    break;
                case '0':
                    std::cout << "Выход из программы.\n";
                    break;
                default:
                    std::cout << "Неверное значение.\n";
                    break;
            }
            if (c != '0') printMenu();
        } while (c != '0');
    }

    void consoleInput() {
        std::cout << "Вводите данные:\n"
                                "Чтобы вернуться в меню, введите \"" <<
kDefaultStopString << "\"\n";
        performTask(std::cin);
    }

    void fileInput() {
        std::string inputFileName;
        std::ifstream infile;
        std::cout << "Введите название файла:\n"
                            "По умолчанию данные читаются из файла \"" <<
kDefaultFileName << "\".\n";
        getline(std::cin, inputFileName);

        if (inputFileName.empty()) {
            inputFileName = kDefaultFileName;
        }

        infile.open(inputFileName);
        if (!infile) {
                 std::cout << "Файла \"" << inputFileName << "\" не
существует.\n";
        } else {
              std::cout << "Чтение данных прекратится на строке \"" <<
kDefaultStopString << "\".\n";
            performTask(infile);
        }

        if (infile.is_open()) {
            infile.close();
        }
    }

    void performTask(std::istream& infile)
    {
```

```
        std::string str;
        std::vector<int> vec;

        while (!infile.eof()) {
            getline(infile, str);
            if (str.empty()) continue;
            if (str == kDefaultStopString) {
                    Logger::instance().log("Встретилась терминальная
строка.\n\n");
                return;
            }

            const char *cstr = str.c_str();
            for (;;) {
                char* pEnd;
                const long i = std::strtol(cstr, &pEnd, 10);
                if (cstr == pEnd) break;
                cstr = pEnd;
                vec.push_back(i);
            }

            Logger::instance().log("\nВведён "); printVector(vec);
            auto vec2 = vec;
            auto vec3 = vec;
               Logger::instance().log("\nСортировка пузырьком/выбором:\
n");
            bubbleSelectionSort(vec);
            Logger::instance().log("\nШейкерная сортировка:\n");
            shakerSort(vec2);
            Logger::instance().log("\nstd::sort:\n");
            std::sort(vec3.begin(), vec3.end());
            printVector(vec);
            vec.clear();
            vec2.clear();
            vec3.clear();
        }
    }

    int main() {
        try {
            menu();
        } catch (std::exception&) {
            std::cout << "menu(): Exception caught\n";
        }
        return 0;
    }
```

Название файла: Logger.h

```
#ifndef ALG_LAB3_LOGGER_H
#define ALG_LAB3_LOGGER_H

#include <iostream>
#include <fstream>
#include <string>
#include <ctime>
```

```
class Logger {
public:
    static Logger& instance();
     void log(const std::string& str, bool toConsole = true, bool
toFile = true);
       void logNodeOperatorEquals(const std::string& first, const
std::string& second, bool res);
private:
    Logger();
    ~Logger();
    Logger(const Logger&) = delete;
    Logger(Logger&&) = delete;
    Logger& operator=(const Logger&) = delete;
    Logger& operator=(Logger&&) = delete;

    static Logger logger;
    std::ofstream stream;
};


#endif //ALG_LAB3_LOGGER_H
```

Название файла: Logger.cpp

```
#include "Logger.h"

Logger::Logger() {
    std::time_t t = std::time(nullptr);
    std::tm* now = std::localtime(&t);
    char logFileName[32];
    strftime(logFileName, 32, "log_%F_%T.txt", now);
    stream.open(logFileName);
}

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

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

void Logger::log(const std::string& str, bool toConsole, bool
toFile) {
    if (toConsole) std::cout << str << '\n';
    if (toFile) stream << str << '\n';
}
```