Лабораторная работа №1
«СОРТИРОВКИ»

по дисциплине "Методы программирования"

Выполнил Ковалев Даниил
СКБ171, вариант 12
МИЭМ НИУ ВШЭ

В данной лабораторной работе реализованы следующие алгоритмы сортировки:

- Шейкер (shaker). Асимптотическая сложность $O(n^2)$.
- Быстрая (quick). Асимптотическая сложность в среднем $O(n \log n)$, но в худшем случае может достигать $O(n^2)$.
- Пирамидальная (heap). Асимптотическая сложность в худшем, среднем и лучшем случаях совпадает и равна $O(n \log n)$

Алгоритмы применяются для сортировки массива элементов Entry — данных о футбольных командах, принимающих участие в чемпионате страны по футболу: страна, название клуба, город, год, ФИО главного тренера (сравнение по полям — год, страна количество набранных очков (по убыванию), название клуба).

Считывание данных в программе происходит либо из csv-файла с заголовком country;city;club;trainer;year;score, либо из базы данных SQLite с таблицей entries и полями country (TEXT), city (TEXT), club (TEXT), trainer (TEXT), year (INTEGER), score (INTEGER).

Схема запуска: запускать нужно python run.py, он запустит генерацию массива из 100000 элементов типа Entry и положит их базу данных data.sqlite:

```
../generate_data/generate_data --size=100000 --output=data.sqlite
```

После этого необходимо запустить тестирование алгоритмов сортировок, взяв в качестве данных сгенерированный data.sqlite; результаты тестирования положить в results.csv. Размеры массивов, на которых тестируются сортировки, берутся из sizes.txt, и являются равномерно распределенными на логарифмической оси от 100 до 100000.

```
./1_test_sort --input=data.sqlite --sizes=sizes.txt --output=results.
    csv 2> test_sort.log
```

После этого средствами пакета matplotlib для python на основе results.csv строится график зависимости времени выполнения сортировки от размера массива данных в логарифмических осях.

Структура проекта:

- 3rd_party:
  - SQLiteCpp — библиотека для работы с БД SQLite
- entry:
  - CMakeLists.txt
  - entry.h — файл с описанием класса Entry
  - entry.cpp — файл с реализацией перегрузок операций ==, !=, <, >, <=, >=.
- generate_data:
  - CMakeLists.txt
  - geography.h — файл с данными о странах и городах

- names.h — файл с данными об именах и фамилиях
- teams.h — файл с данными о названиях футбольных команд
- main.cpp — файл с реализацией генерации данных

helpers:
- CMakeLists.txt
- io_operations.h — файл с описанием функций ввода-вывода
- io_operations.cpp — файл с реализацией функций ввода-вывода

lab1:
- CMakeLists.txt
- heap_sort.h — файл с реализацией пирамидальной сортировки
- quick_sort.h — файл с реализацией быстрой сортировки
- shaker_sort.h — файл с реализацией шейкер-сортировки
- main.cpp — файл с реализацией тестирования сортировок
- run.py — скрипт запуска генерации данных и тестирования сортировок

CMakeLists.txt — общий конфигурационный cmake-файл проекта

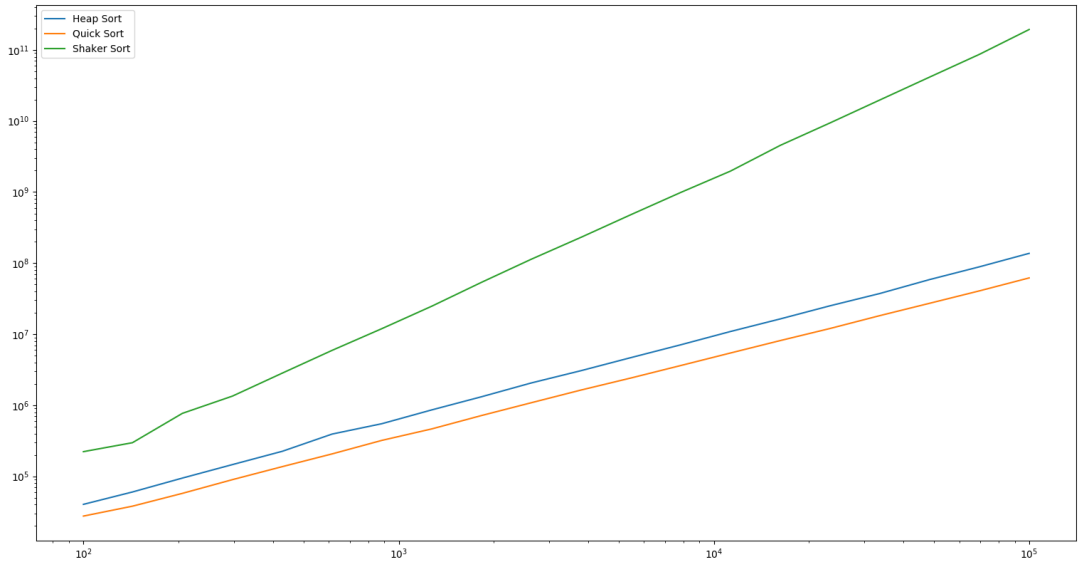
Приведем таблицу, построенную по файлу results.csv. В ней записано время работы каждого из трех алгоритмов сортировки (шейкер, пирамидальной, быстрой) в наносекундах на диапазонах данных размерами от 100 до 100000.

	100	143	206	297	428	615	885
Шейкер	222499	297264	769445	1346166	2841389	5923193	11981680
Пирамидальная	40360	60209	94515	146335	225516	393310	550620
Быстрая	27589	38012	57599	89850	137150	206849	320721

1274	1832	2636	3792	5455	7847	11288
24787447	53710346	113572590	231177805	480736154	986573603	1971329536
860967	1317740	2061820	3070058	4682380	7091648	10910896
465071	717280	1086288	1638973	2418930	3621418	5437617

16237	23357	33598	48329	69519	100000
4536980921	9389715078	19740739039	41419599699	86528260869	193570843468
16402030	25090502	37363043	58668761	88663368	137202619
8130199	12026268	18245677	27212040	40671114	61903511

Из полученного графика видно, что порядок времени работы пирамидальной и быстрой сортировки совпадает и отличается константой (у пирамидальной она больше), а порядок времени работы шейкер-сортировки больше. Это согласуется с теоретическими данными.



Листинги с исходным кодом всех файлов расположены на следующих страницах отчета.

Листинг 1: lab1/CMakeLists.txt

```
set(PROJECT_NAME 1_test_sort)

project(${PROJECT_NAME} LANGUAGES CXX)

set(SOURCES main.cpp)
set(HEADERS heap_sort.h
            quick_sort.h
            shaker_sort.h)

add_executable(${PROJECT_NAME} ${SOURCES} ${HEADERS})

target_include_directories(${PROJECT_NAME} PRIVATE ${Helpers_INCLUDE_DIR} ${
    Entry_INCLUDE_DIR})
target_link_libraries(${PROJECT_NAME} PRIVATE entry)
target_link_libraries(${PROJECT_NAME} PRIVATE helpers)
target_link_libraries(${PROJECT_NAME} PRIVATE Boost::program_options)

file(COPY run.py DESTINATION ${PROJECT_BINARY_DIR})
```

Листинг 2: lab1/heap_sort.h

```
/**
 * @file
 * @brief Заголовочный файл, содержащий реализацию пирамидальной сортировки
 * @date Январь 2020
*/
#ifndef HEAP_SORT_H
#define HEAP_SORT_H

#include <iterator>
#include <functional>

namespace my
{

/** Реализует пирамидальную сортировку диапазона элементов
 * @tparam Iterator Iterator тип, удовлетворяющий C++ named requirement
 * ValueSwappable и LegacyRandomAccessIterator
 * @tparam Comparator тип, удовлетворяющий C++ named requirement Compare
 * @param[in,out] begin,end итераторы, указывающие на диапазон, который
 * требуется отсортировать
 * @param cmp компаратор: возвращает `true`, если его первый аргумент должен стоять
 * в отсортированном диапазоне строго левее второго, `false` иначе
*/
template<typename Iterator, typename Comparator>
void heap_sort(Iterator begin, Iterator end, Comparator cmp)
{
    using diff_t = typename std::iterator_traits<Iterator>::difference_type;
    diff_t size = std::distance(begin, end);
    if (size <= 1)
        return;
    diff_t heap_size = size;

    auto sift_down = [&cmp, &begin, &heap_size](Iterator it)
    {
        diff_t i = std::distance(begin, it);
        while (2 * i + 1 < heap_size)
        {
            diff_t left = 2 * i + 1;
            diff_t right = 2 * i + 2;
            diff_t j = left;
            if (right < heap_size && cmp(*std::next(begin, left), *std::next(begin,
    right)))
                j = right;
            Iterator it = std::next(begin, i);
            Iterator jt = std::next(begin, j);
            if (!cmp(*it, *jt))
```

```
46                  break;
47              std::iter_swap(it, jt);
48              i = std::distance(begin, jt);
49          }
50      };
51
52      for (Iterator it = std::next(begin, size / 2); it != std::prev(begin); --it)
53          sift_down(it);
54
55      for (diff_t i = 0; i < size; ++i)
56      {
57          std::iter_swap(begin, std::prev(end, i + 1));
58          --heap_size;
59          sift_down(begin);
60      }
61  }
62
63  /** Реализует пирамидальную сортировку диапазона элементов по возрастанию
64   * @tparam Iterator Iterator тип, удовлетворяющий C++ named requirement
65   * ValueSwappable и LegacyRandomAccessIterator
66   * @param[in,out] begin,end итераторы, указывающие на диапазон, который
67   * требуется отсортировать
68  */
69  template<typename Iterator>
70  void heap_sort(Iterator begin, Iterator end)
71  {
72      using elem_type = typename std::iterator_traits<Iterator>::value_type;
73      heap_sort(begin, end, std::less<elem_type>());
74  }
75
76  } // namespace my
77
78  #endif // HEAP_SORT_H
```

Листинг 3: lab1/quick_sort.h

```
1   /**
2    * @file
3    * @brief Заголовочный файл, содержащий реализацию быстрой сортировки
4    * @date Январь 2020
5   */
6   #ifndef QUICK_SORT_H
7   #define QUICK_SORT_H
8
9   #include <iterator>
10  #include <stack>
11  #include <functional>
12
13  namespace my
14  {
15
16  /** Реализует быструю сортировку диапазона элементов
17   * @tparam Iterator Iterator тип, удовлетворяющий C++ named requirement
18   * ValueSwappable и LegacyRandomAccessIterator
19   * @tparam Comparator тип, удовлетворяющий C++ named requirement Compare
20   * @param[in,out] begin,end итераторы, указывающие на диапазон, который
21   * требуется отсортировать
22   * @param cmp компаратор: возвращает `true`, если его первый аргумент должен стоять
23   * в отсортированном диапазоне строго левее второго, `false` иначе
24  */
25  template<typename Iterator, typename Comparator>
26  void quick_sort(Iterator begin, Iterator end, Comparator cmp)
27  {
28      using diff_t = typename std::iterator_traits<Iterator>::difference_type;
29      auto partition = [&cmp](Iterator begin, Iterator end)
30      {
31          Iterator left = begin, right = end;
32          diff_t size = std::distance(begin, end);
33          auto pivot = *std::next(left, size / 2);
```

```
34          while (std::distance(left, right) >= 0)
35          {
36              while (cmp(*left, pivot))
37                  ++left;
38              while (cmp(pivot, *right))
39                  --right;
40              if (std::distance(left, right) >= 0)
41                  std::iter_swap(left++, right--);
42          }
43          return std::make_pair(left, right);
44      };
45
46      --end;
47      std::stack<std::pair<Iterator, Iterator>> operations;
48      operations.emplace(begin, end);
49      while (!operations.empty())
50      {
51          auto [left, right] = operations.top();
52          operations.pop();
53          auto [i, j] = partition(left, right);
54          if (std::distance(left, j) > 0)
55              operations.emplace(left, j);
56          if (std::distance(i, right) > 0)
57              operations.emplace(i, right);
58      }
59  }
60
61  /** Реализует быструю сортировку диапазона элементов по возрастанию
62   * @tparam Iterator Iterator тип, удовлетворяющий C++ named requirement
63   * ValueSwappable и LegacyRandomAccessIterator
64   * @param[in,out] begin,end итераторы, указывающие на диапазон, который
65   * требуется отсортировать
66  */
67  template<typename Iterator>
68  void quick_sort(Iterator begin, Iterator end)
69  {
70      using elem_type = typename std::iterator_traits<Iterator>::value_type;
71      quick_sort(begin, end, std::less<elem_type>());
72  }
73
74  } // namespace my
75
76  #endif // QUICK_SORT_H
```

Листинг 4: lab1/shaker_sort.h

```
1   /**
2    * @file
3    * @brief Заголовочный файл, содержащий реализацию шейкер-сортировки
4    * @date Январь 2020
5   */
6   #ifndef SHAKER_SORT_H
7   #define SHAKER_SORT_H
8
9   #include <iterator>
10  #include <functional>
11
12  namespace my
13  {
14
15  /** Реализует шейкер-сортировку диапазона элементов
16   * @tparam Iterator Iterator тип, удовлетворяющий C++ named requirement
17   * ValueSwappable и LegacyRandomAccessIterator
18   * @tparam Comparator тип, удовлетворяющий C++ named requirement Compare
19   * @param[in,out] begin,end итераторы, указывающие на диапазон, который
20   * требуется отсортировать
21   * @param cmp компаратор: возвращает `true`, если его первый аргумент должен стоять
22   * в отсортированном диапазоне строго левее второго, `false` иначе
23  */
```

```
24 template<typename Iterator, typename Comparator>
25 void shaker_sort(Iterator begin, Iterator end, Comparator cmp)
26 {
27     using diff_t = typename std::iterator_traits<Iterator>::difference_type;
28     diff_t size = std::distance(begin, end);
29     if (size <= 1)
30         return;
31     diff_t left = 0, right = size;
32     bool exit_loop;
33     auto action = [&begin, &cmp, &exit_loop](diff_t j)
34     {
35         auto& first = *std::next(begin, j);
36         auto& second = *std::next(begin, j + 1);
37         if (!cmp(first, second))
38         {
39             std::swap(first, second);
40             exit_loop = false;
41         }
42     };
43     for (;;)
44     {
45         exit_loop = true;
46         for (diff_t j = left; j < right - 1; ++j)
47             action(j);
48         --right;
49         for (diff_t j = right - 1; j >= left; --j)
50             action(j);
51         ++left;
52         if (exit_loop)
53             break;
54     }
55 }
56
57 /** Реализует шейкер-сортировку диапазона элементов по возрастанию
58  * @tparam Iterator Iterator тип, удовлетворяющий C++ named requirement
59  * ValueSwappable и LegacyRandomAccessIterator
60  * @param[in,out] begin,end итераторы, указывающие на диапазон, который
61  * требуется отсортировать
62 */
63 template<typename Iterator>
64 void shaker_sort(Iterator begin, Iterator end)
65 {
66     using elem_type = typename std::iterator_traits<Iterator>::value_type;
67     shaker_sort(begin, end, std::less<elem_type>());
68 }
69
70 } // namespace my
71
72 #endif // SHAKER_SORT_H
```

Листинг 5: lab1/main.cpp

```
1  #include "entry.h"
2  #include "io_operations.h"
3  #include "heap_sort.h"
4  #include "quick_sort.h"
5  #include "shaker_sort.h"
6  #include <boost/program_options.hpp>
7  #include <algorithm>
8  #include <chrono>
9  #include <fstream>
10 #include <functional>
11 #include <iostream>
12 #include <map>
13 #include <stdexcept>
14 #include <string>
15 #include <vector>
16
17 using ArraySize = std::size_t;
```

```
18 using SortName = std::string;
19 using Time = std::int64_t;
20 using Data = std::vector<Entry>;
21 using Iterator = Data::iterator;
22 using SizeToTime = std::map<ArraySize, Time>;
23 using TestResult = std::map<SortName, SizeToTime>;
24
25 SizeToTime test_sort(const std::function<void(Iterator, Iterator)>& sort_function,
26                      const Data& data, const std::vector<ArraySize>& sizes)
27 {
28     SizeToTime answer;
29     using namespace std::chrono;
30     for (ArraySize size : sizes)
31     {
32         size = std::min(size, data.size());
33         if (answer.contains(size))
34             continue;
35         Data data_copy(data.begin(), std::next(data.begin(), static_cast<std::
    ptrdiff_t>(size)));
36         time_point<high_resolution_clock> start = high_resolution_clock::now();
37         sort_function(data_copy.begin(), data_copy.end());
38         time_point<high_resolution_clock> end = high_resolution_clock::now();
39         answer[size] = duration_cast<std::chrono::nanoseconds>(end - start).count();
40 #ifndef DNDEBUG
41         if (!std::is_sorted(data_copy.begin(), data_copy.end()))
42             throw std::runtime_error("Wrong sort algorithm");
43 #endif
44     }
45     return answer;
46 }
47
48 TestResult test_all(const Data& data, const std::vector<ArraySize>& sizes)
49 {
50     std::map<SortName, std::function<void(Iterator, Iterator)>> name_to_function =
51     {
52         { "Quick Sort", my::quick_sort<Iterator> },
53         { "Heap Sort", my::heap_sort<Iterator> },
54         { "Shaker Sort", my::shaker_sort<Iterator> }
55     };
56     TestResult answer;
57     for (auto& [name, function] : name_to_function)
58     {
59         std::cerr << "Testing " << name << "..." << std::endl;
60         answer.emplace(name, test_sort(function, data, sizes));
61         std::cerr << "Done!" << std::endl;
62     }
63     return answer;
64 }
65
66 int main(int argc, char* argv[])
67 {
68     std::ios::sync_with_stdio(false);
69     std::cin.tie(nullptr);
70
71     namespace po = boost::program_options;
72     po::options_description desc("Allowed options");
73     desc.add_options()
74         ("help,H", "Print this message")
75         ("sizes,S", po::value<std::string>()->required(), "Text file with data sizes
     to be testes in the following format:\n"
76                                                           "size_0 size_1 size_2 ...
    size_n")
77
78         ("input,I", po::value<std::string>()->required(), "File (csv or sqlite) with
     football clubs data. Format:\n"
79                                                           "* if csv: country;club;
    city;trainer;year;score\n"
80                                                           "* if sqlite: table '
    entries' with columns 'country', 'club', 'city', 'trainer', 'year', 'score'")
81         ("format,F", po::value<std::string>(), "Input file format (csv or sqlite)")
```

```
82          ("output,O", po::value<std::string>()->required(), "csv file to write test
    results, the format is:\n"
83                                                              "sort_name;
    result_for_size_0;...;result_for_size_n")
84          ;
85
86      po::variables_map vm;
87      try
88      {
89          po::store(parse_command_line(argc, argv, desc), vm);
90          if (vm.contains("help"))
91          {
92              std::cout << desc << "\n";
93              return 0;
94          }
95          po::notify(vm);
96      }
97      catch (const po::error& error)
98      {
99          std::cerr << "Error while parsing command-line arguments: "
100                   << error.what() << "\nPlease use --help to see help message\n";
101         return 1;
102     }
103
104     std::string input_filename = vm["input"].as<std::string>();
105     std::string sizes_filename = vm["sizes"].as<std::string>();
106     std::string output_filename = vm["output"].as<std::string>();
107     std::string format;
108     if (vm.contains("format"))
109     {
110         format = vm["format"].as<std::string>();
111     }
112     else
113     {
114         if (input_filename.ends_with(".csv"))
115         {
116             format = "csv";
117         }
118         else if (input_filename.ends_with(".sqlite"))
119         {
120             format = "sqlite";
121         }
122         else
123         {
124             std::cerr << "Invalid format. Please either specify format manually with
     "
125                       "--format or use --input with extension .csv or .sqlite.\n"
126                       "Please use --help to see detailed help message";
127         }
128     }
129
130     if (format != "csv" && format != "sqlite")
131     {
132         std::cerr << "Invalid format. Please use --help see help message\n";
133         return 1;
134     }
135
136     std::cerr << "Reading data..." << std::endl;
137     Data data;
138     if (format == "csv")
139         data = read_data_from_csv(input_filename);
140     else if (format == "sqlite")
141         data = read_data_from_sqlite(input_filename);
142     std::vector<ArraySize> sizes = read_sizes(sizes_filename);
143     shrink_sizes(sizes, data.size());
144     std::cerr << "Done!" << std::endl;
145
146     // csv header
147     std::ofstream output(output_filename);
148     output << "name";
```

```
149     for (ArraySize size : sizes)
150         output << ';' << size;
151     output << '\n';
152 
153     TestResult results = test_all(data, sizes);
154     for (auto& [name, timings] : results)
155     {
156         std::cerr << std::endl << "Algorithm: " << name << std::endl;
157         for (auto [size, time] : timings)
158             std::cerr << size << ": " << time << std::endl;
159         print_timings_csv_line(output, name, timings);
160     }
161 
162     return 0;
163 }
```

Листинг 6: lab1/run.py

```
1  import subprocess
2  import os
3  import numpy as np
4  import pandas as pd
5  from matplotlib import pyplot as plt
6 
7  try:
8      print("Generating data...")
9      if not os.path.isfile("data.csv"):
10         subprocess.call("../generate_data/generate_data --size=100000 --output=data.
       sqlite", shell=True)
11     print("Done!")
12 
13     if not os.path.isfile("sizes.txt"):
14         f = open("sizes.txt", "w")
15         f.write(" ".join(np.logspace(2, 5, 20).astype(int).astype(str)) + "\n")
16         f.close()
17 
18     print("Running tests...")
19     if not os.path.isfile("results.csv"):
20         subprocess.call("./1_test_sort --input=data.sqlite --sizes=sizes.txt --
       output=results.csv 2> test_sort.log", shell=True)
21     print("Done!")
22 
23     raw = pd.read_csv("results.csv", sep=';')
24     data = dict()
25     columns = list(raw.columns[1:].astype(int))
26     for i in range(raw.shape[0]):
27         data[raw.iloc[i]["name"]] = list(raw.iloc[i][1:])
28     plt.figure(figsize=(12, 8))
29     for name, timings in data.items():
30         plt.plot(columns, timings, label=name)
31         plt.xscale('log')
32         plt.yscale('log')
33         plt.legend()
34     plt.show()
35 except Exception as e:
36     print(e)
```

Листинг 7: entry/CMakeLists.txt

```
1 set(LIBRARY_NAME entry)
2 
3 project(${LIBRARY_NAME} LANGUAGES CXX)
4 
5 set(SOURCES entry.cpp)
6 set(HEADERS entry.h)
7 
8 add_library(${LIBRARY_NAME} ${SOURCES} ${HEADERS})
```

```
9
10 target_include_directories(${LIBRARY_NAME} PUBLIC ${SQLiteCpp_INCLUDE_DIR})
11 target_link_libraries(${LIBRARY_NAME} PRIVATE SQLiteCpp sqlite3)
```

Листинг 8: entry/entry.h

```
1  /**
2   * @file
3   * @brief Заголовочный файл, содержащий описание класса Entry
4   * @date Январь 2020
5   */
6  #ifndef ENTRY_H
7  #define ENTRY_H
8
9  #include "SQLiteCpp/SQLiteCpp.h"
10 #include <ostream>
11 #include <string>
12 #include <utility>
13
14 /**
15  * @class Entry
16  * @brief Класс, содержащий описание футбольной команды
17  */
18 class Entry
19 {
20 public:
21     using Country = std::string;
22     using City = std::string;
23     using Club = std::string;
24     using Trainer = std::string;
25     using Year = int;
26     using Score = int;
27
28     Entry() = delete;
29
30     Entry(Country country, City city, Club club,
31           Trainer trainer, Year year, Score score)
32         : m_country(std::move(country))
33         , m_city(std::move(city))
34         , m_club(std::move(club))
35         , m_trainer(std::move(trainer))
36         , m_year(year), m_score(score)
37     {}
38
39     Entry(const Entry&) = default;
40     Entry& operator=(const Entry&) = default;
41     Entry(Entry&&) = default;
42     Entry& operator=(Entry&&) = default;
43
44     [[nodiscard]] Country country() const { return m_country; }
45     [[nodiscard]] City city() const { return m_city; }
46     [[nodiscard]] Club club() const { return m_club; }
47     [[nodiscard]] Trainer trainer() const { return m_trainer; }
48     [[nodiscard]] Year year() const { return m_year; }
49     [[nodiscard]] Score score() const { return m_score; }
50
51     /**
52      * Выводит данные о классе в заданный поток вывода в формате csv:
53      * country;city;club;trainer;year;score
54      * @param[out] stream поток вывода, в который производится вывод
55      * @param[in] sep разделитель, использующийся в формате csv
56      */
57     void to_csv(std::ostream& stream, char sep = ';') const;
58
59     inline static const std::string table_name = "entries";
60     /**
61      * Выводит данные о классе в заданный поток вывода в базу данных SQLite
62      * @param[out] db объект базы данных, в которую нужно произвести запись
63      * @param[in] table имя таблицы в базе `db`, в которую производится запись;
```

```
64          * должна иметь поля `country (TEXT), city (TEXT), club (TEXT), `
65          * `trainer (TEXT), year (INTEGER), score (INTEGER)`
66          */
67         void to_sqlite(SQLite::Database& db, const std::string& table = table_name)
            const;
68
69         friend bool operator==(const Entry& lhs, const Entry& rhs);
70         friend bool operator!=(const Entry& lhs, const Entry& rhs);
71         friend bool operator<(const Entry& lhs, const Entry& rhs);
72         friend bool operator>(const Entry& lhs, const Entry& rhs);
73         friend bool operator<=(const Entry& lhs, const Entry& rhs);
74         friend bool operator>=(const Entry& lhs, const Entry& rhs);
75         friend std::ostream& operator<<(std::ostream& stream, const Entry& entry);
76
77         operator Club() { return m_club; }
78     private:
79         Country m_country;
80         City m_city;
81         Club m_club;
82         Trainer m_trainer;
83         Year m_year;
84         Score m_score;
85     };
86
87     /**
88      * Создает объект класса `Entry` из строки в формате csv
89      * @param[in] csv_line строка в формате csv со следующими столбцами:
90      * country;city;club;trainer;year;score
91      * @param[in] sep разделитель, использующийся в формате csv
92      * @return созданный по строке в формате csv объект класса `Entry`
93      */
94     Entry from_csv(const std::string& csv_line, char sep = ';');
95
96     /**
97      * Создает объект класса `Entry` по данным из БД SQLite
98      * @param[in] query сформированный SQL-запрос; будут использованы поля:
99      * country (TEXT), city (TEXT), club (TEXT), trainer (TEXT), `
100     * `year (INTEGER), score (INTEGER)`
101     * @return созданный по данным из БД объект класса `Entry`
102     */
103    Entry from_sqlite(SQLite::Statement& query);
104
105    #endif // ENTRY_H
```

Листинг 9: entry/entry.cpp

```
1   #include "entry.h"
2   #include <ostream>
3   #include <sstream>
4   #include <stdexcept>
5   #include <tuple>
6
7   bool operator==(const Entry& lhs, const Entry& rhs)
8   {
9       return std::tie(lhs.m_club, lhs.m_year, lhs.m_country, lhs.m_score) ==
10              std::tie(rhs.m_club, rhs.m_year, rhs.m_country, rhs.m_score);
11  }
12
13  bool operator!=(const Entry& lhs, const Entry& rhs)
14  {
15      return !(lhs == rhs);
16  }
17
18  bool operator<(const Entry& lhs, const Entry& rhs)
19  {
20      double lhs_reversed_score = 1. / lhs.m_score;
21      double rhs_reversed_score = 1. / rhs.m_score;
22      return std::tie(lhs.m_club, lhs.m_year, lhs.m_country, lhs_reversed_score) <
23              std::tie(rhs.m_club, rhs.m_year, rhs.m_country, rhs_reversed_score);
```

```
24 }
25
26 bool operator>(const Entry& lhs, const Entry& rhs)
27 {
28     return (rhs < lhs);
29 }
30
31 bool operator<=(const Entry& lhs, const Entry& rhs)
32 {
33     return !(lhs > rhs);
34 }
35 bool operator>=(const Entry& lhs, const Entry& rhs)
36 {
37     return !(lhs < rhs);
38 }
39
40 void Entry::to_csv(std::ostream& stream, char sep) const
41 {
42     stream << m_country << sep
43            << m_city << sep
44            << m_club << sep
45            << m_trainer << sep
46            << m_year << sep
47            << m_score << '\n';
48 }
49
50 void Entry::to_sqlite(SQLite::Database &db, const std::string& table) const
51 {
52     db.exec("INSERT INTO " + table + " "
53             "(country, city, club, trainer, year, score)"
54             "VALUES ("
55             + "\"" + m_country        + "\", "
56             + "\"" + m_city           + "\", "
57             + "\"" + m_club           + "\", "
58             + "\"" + m_trainer        + "\", "
59             + std::to_string(m_year)  + ", "
60             + std::to_string(m_score) + ")");
61 }
62
63 Entry from_csv(const std::string& csv_line, char sep)
64 {
65     Entry::Country country;
66     Entry::City city;
67     Entry::Club club;
68     Entry::Trainer trainer;
69     Entry::Year year;
70     Entry::Score score;
71     std::string remaining;
72     std::istringstream input(csv_line);
73     getline(input, country, sep);
74     getline(input, city, sep);
75     getline(input, club, sep);
76     getline(input, trainer, sep);
77     input >> year;
78     if (input.peek() != sep)
79         throw std::runtime_error("Invalid separator or wrong format of year");
80     input.ignore(1);
81     input >> score;
82     getline(input, remaining);
83     if (!remaining.empty())
84         throw std::runtime_error("Invalid csv");
85     return Entry(country, city, club, trainer, year, score);
86 }
87
88 Entry from_sqlite(SQLite::Statement& query)
89 {
90     Entry::Country country = query.getColumn("country");
91     Entry::City city = query.getColumn("city");
92     Entry::Club club = query.getColumn("club");
93     Entry::Trainer trainer = query.getColumn("trainer");
```

```
94      Entry::Year year = query.getColumn("year");
95      Entry::Score score = query.getColumn("score");
96      return Entry(country, city, club, trainer, year, score);
97  }
98
99  std::ostream& operator<<(std::ostream& stream, const Entry& entry)
100 {
101     stream << "country: \"" << entry.m_country
102            << "\"\ncity: \"" << entry.m_city
103            << "\"\nclub: \"" << entry.m_club
104            << "\"\ntrainer: \"" << entry.m_trainer
105            << "\"\nyear: " << entry.m_year
106            << "\nscore: " << entry.m_score << '\n';
107     return stream;
108 }
```

Листинг 10: generate_data/CMakeLists.txt

```
1  set(PROJECT_NAME generate_data)
2
3  project(${PROJECT_NAME} LANGUAGES CXX)
4
5  set(SOURCES main.cpp)
6  set(HEADERS geography.h
7              names.h
8              teams.h)
9
10 add_executable(${PROJECT_NAME} ${SOURCES} ${HEADERS})
11
12 target_include_directories(${PROJECT_NAME} PUBLIC ${Entry_INCLUDE_DIR} ${
       SQLiteCpp_INCLUDE_DIR})
13 target_link_libraries(${PROJECT_NAME} PRIVATE entry Boost::program_options)
```

Листинг 11: generate_data/geography.h

```
1  #ifndef GEOGRAPHY_H
2  #define GEOGRAPHY_H
3
4  #include <string>
5  #include <utility>
6  #include <vector>
7
8  using Country = std::string;
9  using City = std::string;
10
11 static const std::vector<std::pair<Country, City>> cities =
12 {
13     { "Andorra", "les Escaldes" },
14     // ...
15     { "Zimbabwe", "Chitungwiza" }
16 };
17
18 #endif // GEOGRAPHY_H
```

Листинг 12: generate_data/names.h

```
1  #ifndef NAMES_H
2  #define NAMES_H
3
4  #include <string>
5  #include <vector>
6
7  static const std::vector<std::string> first_names =
8  {
9      "Joella",
10     // ...
```

```
    "Jimmie"
};

static const std::vector<std::string> last_names =
{
    "Stahl",
    // ...
    "Adan"
};

#endif // NAMES_H
```

Листинг 13: generate_data/teams.h

```
#ifndef TEAMS_H
#define TEAMS_H

#include <string>
#include <vector>

static const std::vector<std::string> teams =
{
    "KRC Genk",
    // ...
    "Lugano"
};

#endif // TEAMS_H
```

Листинг 14: helpers/CMakeLists.txt

```
set(LIBRARY_NAME helpers)

project(${LIBRARY_NAME} LANGUAGES CXX)

set(SOURCES io_operations.cpp)
set(HEADERS io_operations.h)

add_library(${LIBRARY_NAME} ${SOURCES} ${HEADERS})

target_include_directories(${LIBRARY_NAME} PRIVATE ${Entry_INCLUDE_DIR})
target_link_libraries(${LIBRARY_NAME} PRIVATE entry)
```

Листинг 15: helpers/io_operations.h

```
#include "entry.h"
#include <ostream>
#include <map>
#include <string>
#include <vector>

using ArraySize = std::size_t;
using AlgoName = std::string;
using Time = std::int64_t;
using Data = std::vector<Entry>;
using SizeToTime = std::map<ArraySize, Time>;

std::vector<ArraySize> read_sizes(const std::string& sizes_filename);

void shrink_sizes(std::vector<ArraySize>& sizes, ArraySize max_size);

Data read_data_from_csv(const std::string& csv_filename, char sep = ';');
Data read_data_from_sqlite(const std::string& sqlite_filename);

std::ostream& print_timings_csv_line(std::ostream& output, const AlgoName& name,
                                     const SizeToTime& timings, char sep = ';');
```

Листинг 16: helpers/io_operations.cpp

```cpp
#include "io_operations.h"
#include <algorithm>
#include <fstream>
#include <ostream>
#include <map>
#include <string>
#include <vector>

std::vector<ArraySize> read_sizes(const std::string& sizes_filename)
{
    std::ifstream sizes(sizes_filename);
    if (!sizes.is_open())
        throw std::runtime_error("Unable to open " + sizes_filename);
    std::vector<ArraySize> answer;
    ArraySize current_size;
    while (sizes >> current_size)
        answer.emplace_back(current_size);
    return answer;
}

void shrink_sizes(std::vector<ArraySize>& sizes, ArraySize max_size)
{
    for (ArraySize& size : sizes)
        size = std::min(size, max_size);
    std::sort(sizes.begin(), sizes.end());
    auto last = std::unique(sizes.begin(), sizes.end());
    sizes.resize(static_cast<std::size_t>(last - sizes.begin()));
}

Data read_data_from_csv(const std::string& csv_filename, char sep)
{
    std::ifstream input(csv_filename);
    if (!input.is_open())
        throw std::runtime_error("Unable to open " + csv_filename);
    std::string csv_line;
    std::getline(input, csv_line);
    Data answer;
    while (std::getline(input, csv_line))
        answer.emplace_back(from_csv(csv_line, sep));
    return answer;
}

Data read_data_from_sqlite(const std::string& sqlite_filename)
{
    SQLite::Database db(sqlite_filename);
    SQLite::Statement query(db, "SELECT * FROM " + Entry::table_name);
    Data answer;
    while (query.executeStep())
        answer.emplace_back(from_sqlite(query));
    return answer;
}

std::ostream& print_timings_csv_line(std::ostream& output, const AlgoName& name,
                                     const SizeToTime& timings, char sep)
{
    output << name;
    for (auto [_, time] : timings)
        output << sep << time;
    output << '\n';
    return output;
}
```

Листинг 17: generate_data/main.cpp

```cpp
#include "../entry/entry.h"
#include "geography.h"
#include "names.h"
```

```cpp
#include "teams.h"
#include <iostream>
#include <random>
#include <string>
#include <utility>
#include <vector>

std::pair<Country, City> generate_location(std::mt19937& prng)
{
    std::uniform_int_distribution<std::size_t> dist(0, cities.size() - 1);
    return cities[dist(prng)];
}

std::string generate_team(std::mt19937& prng)
{
    std::uniform_int_distribution<std::size_t> dist(0, teams.size() - 1);
    return teams[dist(prng)];
}

std::string generate_trainer(std::mt19937& prng)
{
    std::uniform_int_distribution<std::size_t> dist_first(0, first_names.size() - 1);
    std::uniform_int_distribution<std::size_t> dist_last(0, last_names.size() - 1);
    return first_names[dist_first(prng)] + " " + last_names[dist_last(prng)];
}

int generate_year(std::mt19937& prng)
{
    return std::uniform_int_distribution<int>(1990, 2020)(prng);
}

int generate_score(std::mt19937& prng)
{
    return std::uniform_int_distribution<int>(0, 100)(prng);
}

int main(int argc, char* argv[])
{
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    if (argc != 2)
    {
        std::cerr << "Invalid arguments. Usage:\n"
                     "./generate_data <data_size>\n"
                     "<data_size> - number of lines in csv\n";
        return 1;
    }

    int size = std::stoi(std::string(argv[1]));
    std::mt19937 prng(std::random_device{}());
    std::cout << "country;city;club;trainer;year;score\n";

    for (int i = 0; i < size; ++i)
    {
        auto [country, city] = generate_location(prng);
        std::string club = generate_team(prng);
        std::string trainer = generate_trainer(prng);
        int year = generate_year(prng);
        int score = generate_score(prng);
        Entry(country, city, club, trainer, year, score).to_csv(std::cout);
    }

    return 0;
}
```

Листинг 18: CMakeLists.txt

```
cmake_minimum_required(VERSION 3.5)

```

```
3  project(programming-techniques LANGUAGES CXX)
4
5  set(CMAKE_CXX_STANDARD 20)
6  set(CMAKE_CXX_STANDARD_REQUIRED ON)
7
8  set(THIRD_PARTY_DIR ${CMAKE_CURRENT_LIST_DIR}/3rd_party)
9
10 set(SQLiteCpp_ROOT_DIR ${THIRD_PARTY_DIR}/SQLiteCpp)
11 set(SQLiteCpp_INCLUDE_DIR ${SQLiteCpp_ROOT_DIR}/include)
12 add_subdirectory(${SQLiteCpp_ROOT_DIR})
13
14 find_package(Boost COMPONENTS COMPONENTS program_options REQUIRED)
15
16 set(Entry_ROOT_DIR ${CMAKE_CURRENT_LIST_DIR}/entry)
17 set(Entry_INCLUDE_DIR ${Entry_ROOT_DIR})
18 add_subdirectory(${Entry_ROOT_DIR})
19
20 set(Helpers_ROOT_DIR ${CMAKE_CURRENT_LIST_DIR}/helpers)
21 set(Helpers_INCLUDE_DIR ${Helpers_ROOT_DIR})
22 add_subdirectory(${Helpers_ROOT_DIR})
23
24 add_subdirectory(generate_data)
25 add_subdirectory(lab1)
26 add_subdirectory(lab2)
```