Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления» (ИУ)

КАФЕДРА «Информационная безопасность» (ИУ8)

Лабораторная работа № 8

ПО КУРСУ

«Алгоритмические языки»

на тему «Контейнеры библиотеки STL. Предикаты в языке Си++»

Студент ИУ8-13 (Группа) В. С. Ажгирей (И. О. Фамилия)

Преподаватель: М. В. Малахов (И.О. Фамилия)

2023

Введение

Цели и задачи работы

Цель работы состоит в овладении навыков владения контейнерами библиотеки STL в языке Си++. Для достижения цели необходимо выполнить следующие задачи:

- изучить необходимые учебные материалы, посвященные контейнерам библиотеки STL языка Си++;

- разработать программы на языке Си++ для решения заданных вариантов заданий;

- отладить программы;

- выполнить решение контрольного примера небольшой размерности с помощью программы и ручной расчет контрольного примера;

- подготовить отчет по лабораторной работе.

Условия для 1 варианта

Часть 1: В ходе лабораторной работы студент производит улучшение своих навыков владения контейнерами библиотеки STL. Лабораторная работа состоит из двух частей (необходимо разработать два экземпляра ПО).

В первой части лабораторной работы необходимо провести вычислительные эксперименты по оценки времени выполнения некоторых операций с контейнерами. Необходимо измерять время выполнения заданных операций для пары контейнеров в соответствии со своим вариантом. Для измерения времени можно использовать класс LogDuration, определенный в файле profile.h. Файл можно скачать по ссылке:

http://student.bmstu.cloud:10288/AlgWeb_2022_1/mydata/ZadForSem/profile.h

В работе число выполняемых вызовов функций (итераций циклов, число итераций может быть порядка: 100000, 1000000, 10000000) студент должен подобрать сам с учетом быстродействия компьютера, таким образом, чтобы время выполнения кода было разумным (интервалы измерялись миллисекундами, десятками или сотнями миллисекунд, секундами, в крайнем случае, десятками секунд). Необходимо произвести одинаковое число аналогичных операций (вызовов функций) с двумя сравниваемыми контейнерами. Измерения проводить отдельно по каждому из типов

операций. Заполнять контейнер, а также задавать элементы для поиска и удаления, когда это требуется, необходимо с помощью генератора псевдослучайных чисел, типы элементов контейнера – целые числа.

1) Сравниваемые контейнеры: vector и list.

   Исследуемые операции:

   1. Добавление элементов в конец контейнера.
   2. Поиск элементов.
   3. Удаление элементов из начала контейнера.

Часть 2: Во второй части лабораторной работы необходимо определить структуру в соответствии с заданным вариантов, создать заданный контейнер переменных этого структурного типа. Выполнить сортировку объектов в контейнере, использую алгоритм sort или для контейнера list метод класса sort. Для сортировки использовать специальную функцию предикат, см. пример ниже сортировки вектора по разным полям. Распечатать контейнер объектов до сортировки и после сортировки.

Ввод исходных данных осуществлять из текстового файла (не менее 8 объектов), тестовый файл создать отдельно в простом текстовом редакторе типа «Блокнот», отдельно обрабатывать ситуацию некорректного заполнения файла. Путь к файлу передавать в виде аргумента командной строки.

2) Контейнер: vector. Структура: Объект- сотрудник (поля: ФИО, дата приема на работу, должность, базовый оклад). Сортировка по: дате приема на работу.

Основная часть

Исходный текст программы Часть 1

Исходный текст файла main.cpp:

```
#include "sources.h"

int main()
{
    int numberIterations = 1000, containerSize = 10000;

    AddingElementsToEnd(containerSize, numberIterations);
    SearchingElements(containerSize);
    RemovingElementsFromBeginning(containerSize);

    return 0;
}
```

Исходный текст файла sources.h:

```
#include "sources.h"

void AddingElementsToEnd(const unsigned int containerSize, const unsigned int
numberIterations)
{
    double summaryTimeVector = 0., summaryTimeList = 0.;

    std::vector<int> vector;
    std::srand(time(0));
    for (size_t i = 0; i < containerSize; ++i)
    {
        vector.push_back(rand());
    }

    std::vector<int> vector1;
    for (size_t i = 0; i < numberIterations; ++i)
    {
        vector1.assign(vector.begin(), vector.end());
        auto start = std::chrono::steady_clock::now();
        vector1.push_back(rand());
        auto end = std::chrono::steady_clock::now();
        auto executionTimeMilliseconds =
std::chrono::duration_cast<std::chrono::microseconds>(end - start);

        summaryTimeVector += executionTimeMilliseconds.count();
    }
```

```
    std::cout << "Average execution time of the operation of adding an element to
the end of the container std::vector in milliseconds: " << summaryTimeVector /
numberIterations << std::endl;

    std::list<int> list;
    std::srand(time(0));
    for (size_t i = 0; i < containerSize; ++i)
    {
        list.push_back(rand());
    }

    std::list<int> list1;
    for (size_t i = 0; i < numberIterations; ++i)
    {
        list1.assign(list.begin(), list.end());
        auto start = std::chrono::steady_clock::now();
        list1.push_back(rand());
        auto end = std::chrono::steady_clock::now();
        auto executionTimeMilliseconds =
std::chrono::duration_cast<std::chrono::microseconds>(end - start);

        summaryTimeList += executionTimeMilliseconds.count();
    }

    std::cout << "Average execution time of the operation of adding an element to
the end of the container std::list in milliseconds: " << summaryTimeList /
numberIterations << std::endl
              << std::endl;
}

void SearchingElements(const unsigned int containerSize)
{
    double summaryTimeVector = 0., summaryTimeList = 0.;

    std::vector<int> vector;
    std::srand(time(0));
    for (size_t i = 0; i < containerSize; ++i)
    {
        vector.push_back(rand());
    }

    for (size_t i = 0; i < containerSize; ++i)
    {
        auto start = std::chrono::steady_clock::now();
        std::find(vector.begin(), vector.end(), vector[i]);
        auto end = std::chrono::steady_clock::now();
        auto executionTimeMilliseconds =
std::chrono::duration_cast<std::chrono::microseconds>(end - start);

        summaryTimeVector += executionTimeMilliseconds.count();
```

```
    }
    std::cout << "Average execution time of the element search operation in the
std container::vector in milliseconds: " << summaryTimeVector / containerSize <<
std::endl;

    std::list<int> list;
    std::srand(time(0));
    for (size_t i = 0; i < containerSize; ++i)
    {
        list.push_back(rand());
    }

    for (size_t i = 0; i < containerSize; ++i)
    {
        auto start = std::chrono::steady_clock::now();
        std::find(list.begin(), list.end(), *std::next(list.begin(), i));
        auto end = std::chrono::steady_clock::now();
        auto executionTimeMilliseconds =
std::chrono::duration_cast<std::chrono::microseconds>(end - start);

        summaryTimeList += executionTimeMilliseconds.count();
    }
    std::cout << "Average execution time of the element search operation in the
std container::list in milliseconds: " << summaryTimeList / containerSize <<
std::endl
              << std::endl;
}

void RemovingElementsFromBeginning(const unsigned int containerSize)
{
    double summaryTimeVector = 0., summaryTimeList = 0.;

    std::vector<int> vector;
    std::srand(time(0));
    for (size_t i = 0; i < containerSize; ++i)
    {
        vector.push_back(rand());
    }

    for (size_t i = 0; i < containerSize; ++i)
    {
        auto start = std::chrono::steady_clock::now();
        vector.erase(vector.begin());
        auto end = std::chrono::steady_clock::now();
        auto executionTimeMilliseconds =
std::chrono::duration_cast<std::chrono::microseconds>(end - start);

        summaryTimeVector += executionTimeMilliseconds.count();
    }
```

```
    std::cout << "Average execution time of the operation of removing an element
from the beginning of the container std::vector in milliseconds: " <<
summaryTimeVector / containerSize << std::endl;

    std::list<int> list;
    std::srand(time(0));
    for (size_t i = 0; i < containerSize; ++i)
    {
        list.push_back(rand());
    }

    for (size_t i = 0; i < containerSize; ++i)
    {
        auto start = std::chrono::steady_clock::now();
        list.pop_front();
        auto end = std::chrono::steady_clock::now();
        auto executionTimeMilliseconds =
std::chrono::duration_cast<std::chrono::microseconds>(end - start);

        summaryTimeList += executionTimeMilliseconds.count();
    }
    std::cout << "Average execution time of the operation of removing an element
from the beginning of the container std::list in milliseconds: " <<
summaryTimeList / containerSize << std::endl
              << std::endl;
}
```

Исходный текст программы Часть 2

Исходный текст файла main.cpp:

```
#include "sources.h"

int main(int argc, char *argv[])
{
    if (argc != 2)
    {
        std::cerr << "Invalid number of arguments";
        return 1;
    }

    std::string dataFilePath = argv[1];
    std::vector<Employee> employersVector = readDataEmployees(dataFilePath);

    std::cout << "Employers-vector before sorting by date of employment:" <<
std::endl
              << std::endl;
    print(employersVector);
```

```
    std::sort(employersVector.begin(), employersVector.end(),
SortingByDateEmployment);

    std::cout << "Employers-vector after sorting by date of employment:" <<
std::endl
              << std::endl;
    print(employersVector);

    return 0;
}
```

Исходный текст файла sources.h:

```
#pragma once
#include <algorithm>
#include <fstream>
#include <iostream>
#include <sstream>
#include <string>
#include <vector>

struct Employee
{
    std::string full_name;
    std::string date_employment;
    std::string post;
    unsigned int base_salary;
};

std::vector<Employee> readDataEmployees(const std::string &dataFilePath);
bool SortingByFullName(const Employee &empl1, const Employee &empl2);
bool SortingByDateEmployment(const Employee &empl1, const Employee &empl2);
bool SortingByBaseSalary(const Employee &empl1, const Employee &empl2);
void print(const std::vector<Employee> &employersVector);
```

Исходный текст файла sources.cpp:

```
#include "sources.h"

std::vector<Employee> readDataEmployees(const std::string &dataFilePath)
{
    std::ifstream dataFile(dataFilePath);

    if (!dataFile)
    {
        std::cerr << "Failed to open file" << std::endl;
    }
```

```
    std::vector<Employee> vector;
    Employee empl;
    std::string base_salary;
    while (dataFile >> empl.full_name >> empl.date_employment >> empl.post >>
base_salary)
    {
        empl.base_salary = (unsigned int)std::stoi(base_salary);

        vector.push_back(empl);
    }

    dataFile.close();

    return vector;
}

bool SortingByFullName(const Employee &empl1, const Employee &empl2)
{
    return empl1.full_name < empl2.full_name;
}

bool SortingByDateEmployment(const Employee &empl1, const Employee &empl2)
{
    std::vector<unsigned short int> dateEmploymentEmpl1;
    std::istringstream stream1(empl1.date_employment);
    std::string token;
    while (std::getline(stream1, token, '/'))
    {
        dateEmploymentEmpl1.insert(dateEmploymentEmpl1.begin(),
std::stoi(token));
    }

    std::vector<unsigned short int> dateEmploymentEmpl2;
    std::istringstream stream2(empl2.date_employment);
    while (std::getline(stream2, token, '/'))
    {
        dateEmploymentEmpl2.insert(dateEmploymentEmpl2.begin(),
std::stoi(token));
    }

    return std::lexicographical_compare(dateEmploymentEmpl1.begin(),
dateEmploymentEmpl1.end(),
                                        dateEmploymentEmpl2.begin(),
dateEmploymentEmpl2.end());
}

bool SortingByBaseSalary(const Employee &empl1, const Employee &empl2)
{
    return empl1.base_salary < empl2.base_salary;
}
```

```
void print(const std::vector<Employee> &employersVector)
{
    for (Employee employee : employersVector)
    {
        std::cout << "Employee's full name: " << employee.full_name << std::endl;
        std::cout << "Date of employment: " << employee.date_employment <<
std::endl;
        std::cout << "Post: " << employee.post << std::endl;
        std::cout << "Base salary: " << employee.base_salary << std::endl;
        std::cout << "##############################################" <<
std::endl;
    }
    std::cout << std::endl
              << std::endl;
}
```

Снимки выполнения работы программы

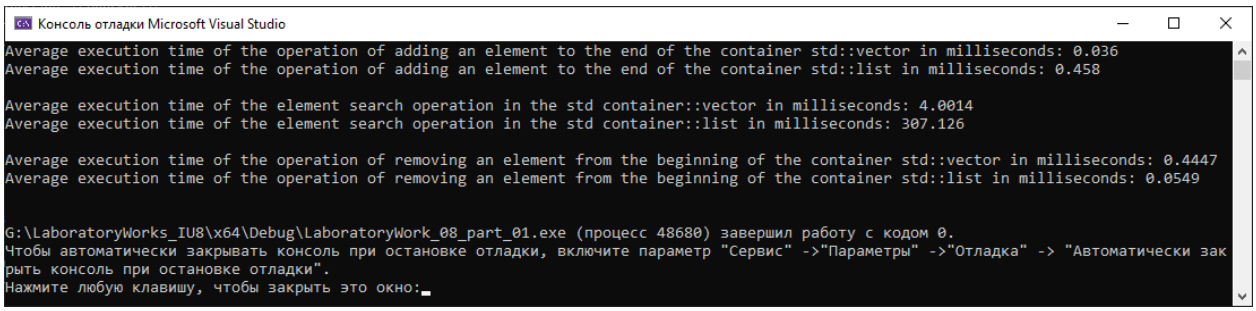
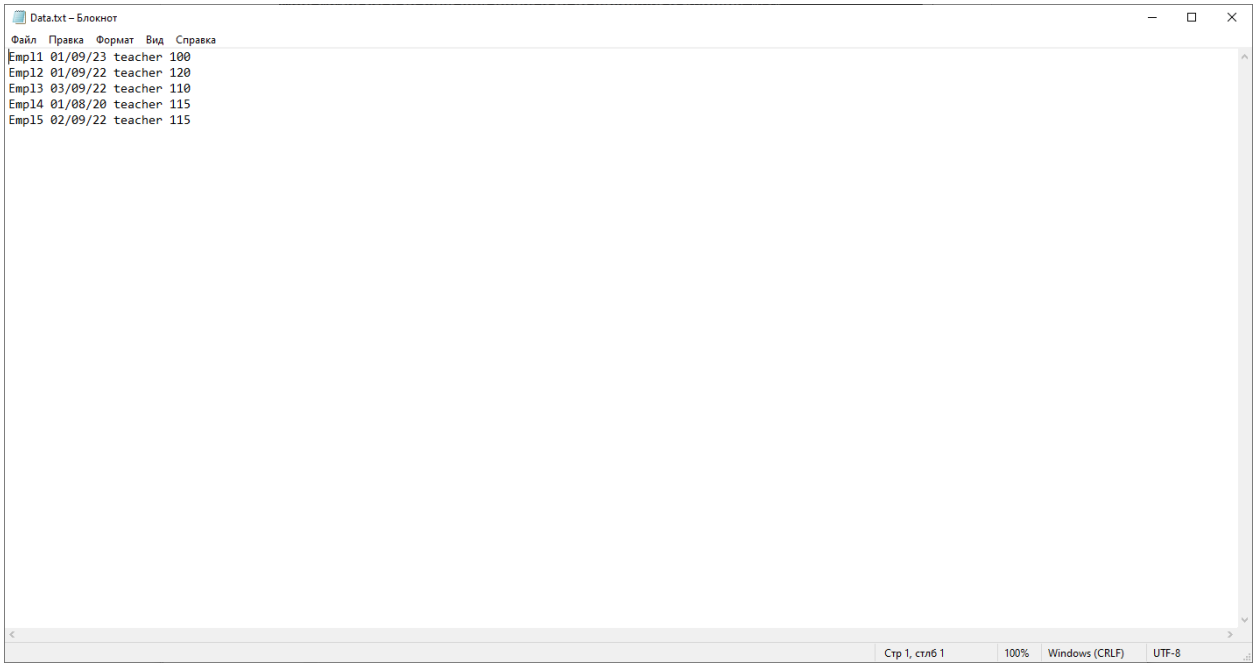


Рисунок 1 – Запуск программы для Части 1 с входными данными numberIterations = 1000, containerSize = 10000.



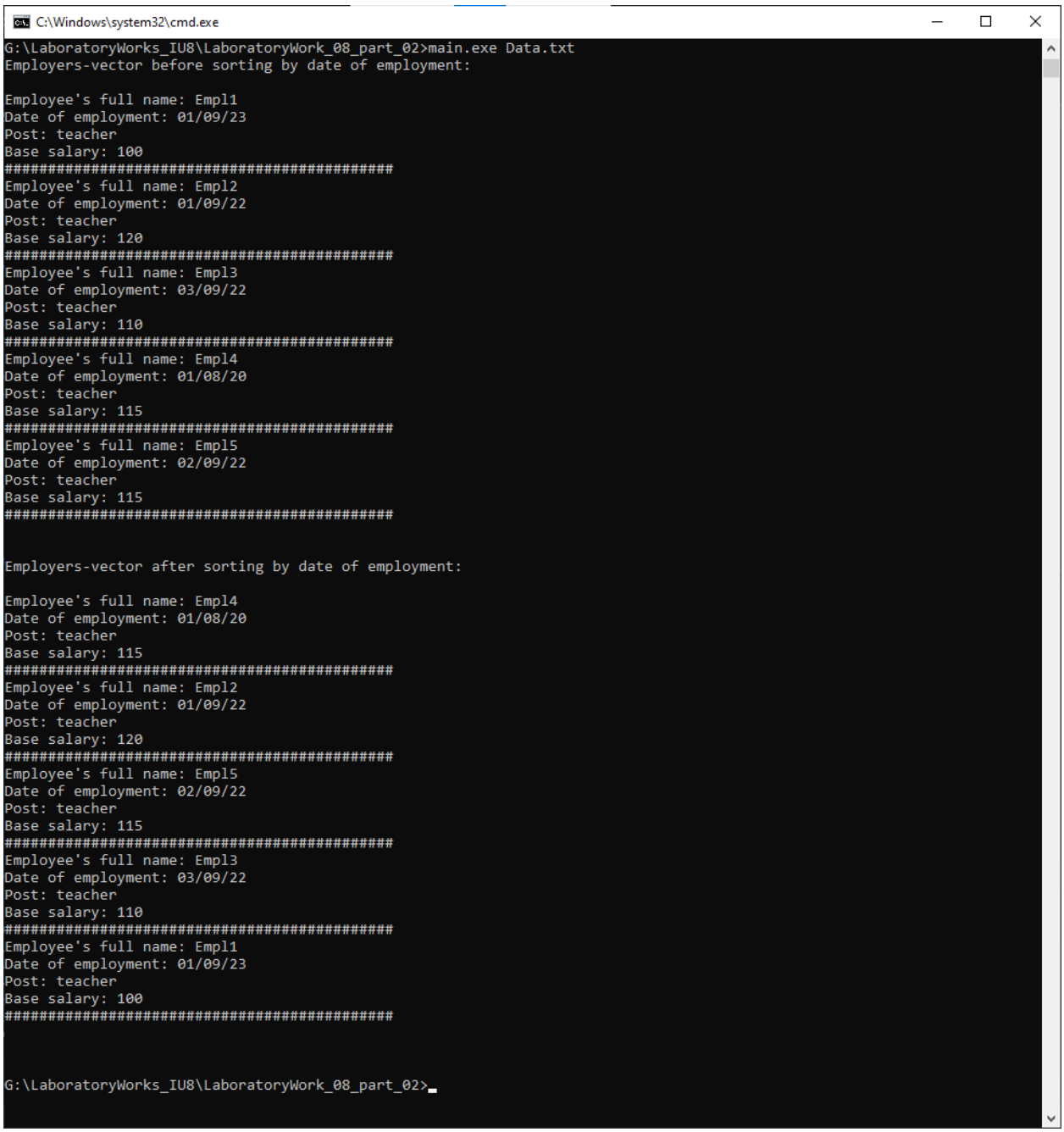


Рисунок 2 – Запуск программы для Части 2 с данными из файла Data.txt .

Заключение

Задачи лабораторной работы были решены, результаты проверены. Изучены на практике функции-предиктаты для сортировки, библиотека algorithm и контейнеры vector и list, библиотека chrono. Проверено время выполнения операций для этих контейнеров.