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 № 7

ПО КУРСУ

«Алгоритмические языки»

на тему «Изучение линейных списков 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 навыками разработки программ на языке Си++, использующих линейные списки, при работе с данными, имеющими динамическую структуру. Для достижения цели необходимо выполнить следующие задачи:

- изучить необходимые учебные материалы, посвященные линейным спискам, имеющим динамическую структуру, 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

В лабораторной работе необходимо организовать список объектов и сортировку списка. Данные списка читаются из файла, путь к файлу указывается в качестве аргумента командной строки. При сортировке элементы списка остаются в оперативной памяти на «своих местах», меняются только значения указателей, связывающие элементы. Вывести на экран список до сортировки и после сортировки.

В работе *ЗАПРЕЩЕНО* использовать списки из библиотеки STL. Список необходимо реализовывать самостоятельно с использованием указателей.

В приложении организовать список объектов и сортировку списка. Для задания строк и массивов использовать шаблоны string и vector.

1) Объект- сотрудник (поля: ФИО, дата приема на работу, должность, базовый оклад). Сортировка по ФИО методом вставки, двухсвязный кольцевой список.

Основная часть

Исходный текст файла main.cpp:

```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];

    List employeesList = readDataEmployees(dataFilePath);

    std::cout << "Before sorting:" << std::endl;
    print(employeesList);

    insertSort(employeesList);

    std::cout << "After sorting:" << std::endl;
    print(employeesList);

    clearList(employeesList);

    return 0;
}
```

Исходный текст файла sources.h:

```cpp
#pragma once
#include <fstream>
#include <iostream>
#include <string>

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

struct Node
{
```

```
    Employee employee;
    Node *pPrev;
    Node *pNext;
};

struct List
{
    Node *pFirst = nullptr;
    Node *pEnd = nullptr;
};

void addBegin(List &list, Node *p);
void addEnd(List &list, Node *p);
void removeNode(List &list, unsigned short index);
Node *extractFront(List &list);
Node *extractBack(List &list);
void print(List &list);
void clearList(List &list);
void insertSort(List &list);
List readDataEmployees(const std::string &dataFilePath);
```

Исходный текст файла sources.cpp:

```
#include "sources.h"

List readDataEmployees(const std::string &dataFilePath)
{
    std::ifstream dataFile(dataFilePath);

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

    List list;
    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);
        Node *p = new Node;
        p->employee = empl;

        addEnd(list, p);
    }

    dataFile.close();
```

```
    return list;
}

void addBegin(List &list, Node *p)
{
    if (list.pFirst == nullptr)
    {
        list.pFirst = list.pEnd = p;
        list.pFirst->pPrev = list.pFirst->pNext = list.pEnd->pPrev = list.pEnd-
>pNext = p;
    }
    else
    {
        p->pNext = list.pFirst;
        p->pPrev = list.pEnd;
        list.pFirst->pPrev = list.pEnd->pNext = p;
        list.pFirst = p;
    }
}

void addEnd(List &list, Node *p)
{
    if (list.pEnd == nullptr)
    {
        list.pFirst = list.pEnd = p;
        list.pFirst->pPrev = list.pFirst->pNext = list.pEnd->pPrev = list.pEnd-
>pNext = p;
    }
    else
    {
        p->pPrev = list.pEnd;
        p->pNext = list.pFirst;
        list.pFirst->pPrev = list.pEnd->pNext = p;
        list.pEnd = p;
    }
}

void removeNode(List &list, unsigned short index)
{
    if (list.pFirst == nullptr)
    {
        std::cerr << "ERROR: List is empty" << std::endl;
        return;
    }
    Node *p = list.pFirst;
    if (p->pPrev == p)
    {
        list.pEnd = list.pFirst = nullptr;
    }
    else
```

```
    {
        for (size_t i = 0; i < index; ++i)
        {
            p = p->pNext;
        }
        p->pPrev->pNext = p->pNext;
        p->pNext->pPrev = p->pPrev;
        if (list.pFirst == p)
            list.pFirst = p->pNext;
        if (list.pEnd == p)
            list.pEnd = p->pPrev;
    }
    delete p;
}

Node *extractFront(List &list)
{
    if (list.pFirst == nullptr)
        return nullptr;
    Node *p = list.pFirst;
    list.pFirst = list.pFirst->pNext;
    if (p == list.pFirst)
        list.pFirst = list.pEnd = nullptr;
    else
    {
        list.pFirst->pPrev = list.pEnd;
        list.pEnd->pNext = list.pFirst;
    }
    return p;
}

Node *extractBack(List &list)
{
    if (list.pEnd == nullptr)
        return nullptr;
    Node *p = list.pEnd;
    list.pEnd = list.pEnd->pPrev;
    if (p == list.pEnd)
        list.pFirst = list.pEnd = nullptr;
    else
    {
        list.pFirst->pPrev = list.pEnd;
        list.pEnd->pNext = list.pFirst;
    }
    return p;
}

void print(List &list)
{
    Node *p = list.pFirst;
```

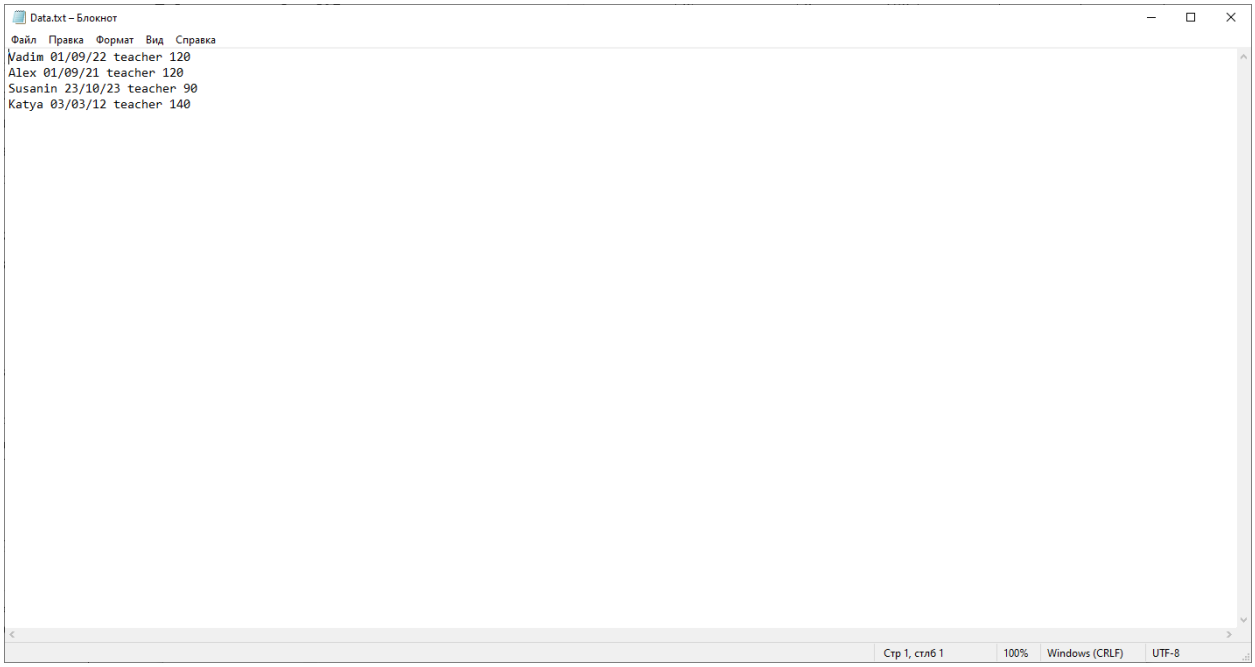
```
    do
    {
        std::cout << "Employee's full name: " << p->employee.full_name <<
std::endl;
        std::cout << "Date of employment: " << p->employee.date_employment <<
std::endl;
        std::cout << "Post: " << p->employee.post << std::endl;
        std::cout << "Base salary: " << p->employee.base_salary << std::endl;
        std::cout << "##############################################" <<
std::endl;

        p = p->pNext;
    } while (p != list.pFirst);
    std::cout << std::endl;
}

void clearList(List &list)
{
    while (list.pFirst != nullptr)
    {
        removeNode(list, 0);
    }
}

void insertSort(List &list)
{
    Node *p = list.pFirst->pNext;
    while (p != list.pFirst)
    {
        Node *currentP = p;
        while (currentP->employee.full_name < currentP->pPrev->employee.full_name
&& currentP != list.pFirst)
        {
            Node *tmp = currentP->pPrev;
            currentP->pPrev->pPrev->pNext = currentP;
            currentP->pPrev = currentP->pPrev->pPrev;
            tmp->pNext = currentP->pNext;
            tmp->pNext->pPrev = tmp;
            tmp->pPrev = currentP;
            currentP->pNext = tmp;
            if (list.pFirst == tmp)
                list.pFirst = currentP;
            if (list.pEnd == currentP)
                list.pEnd = tmp;
            tmp = currentP->pPrev;
        }
        p = p->pNext;
    }
}
```

Снимки выполнения работы программы



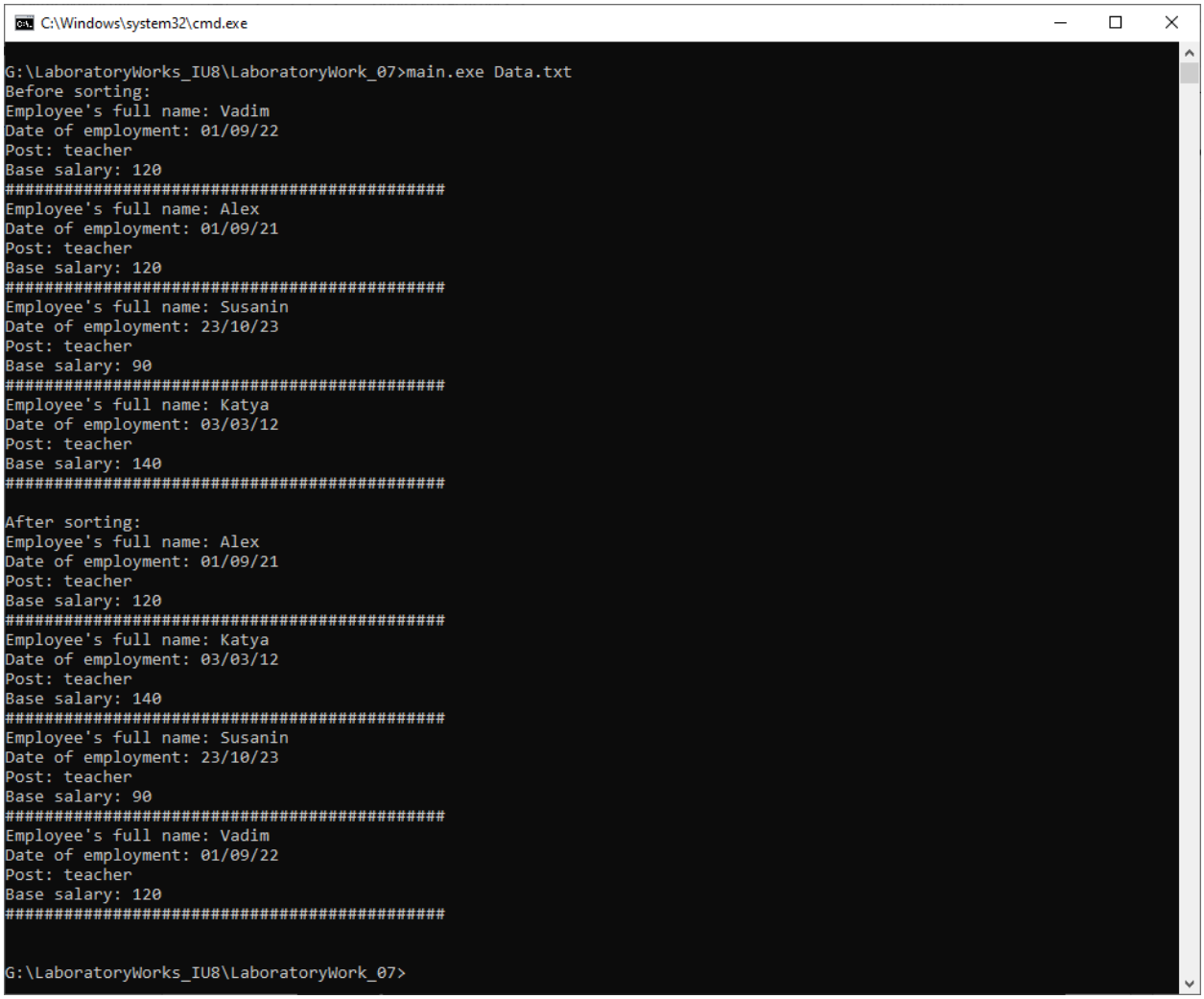


Рисунок 1 – Запуск программы для данный из файла Data.txt .

Заключение

Задачи лабораторной работы были решены, результаты проверены. Изучены на практике алгоритмы сортировки, при которых элементы в оперативной памяти остаются на своих местах, а меняются только указатели, и структура данных «список».