



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)

Домашнее задание № 3
ПО КУРСУ
«Алгоритмические языки»
на тему «Объектно-ориентированное программирование,
библиотеки
классов, исключения, потоковая многозадачность»

Студент

ИУ8-25
(Группа)

В.В.Гоза
(И. О. Фамилия)

Преподаватель:

В. В. Соборова
(И.О. Фамилия)

2022 г.

Условие:

Задание

Разработать программу, работающую с объектами классов. Используя интерфейс командной строки, реализовать следующие режимы работы: «Ввод нового объекта и добавление его в контейнер», «Поиск объекта в контейнере по значениям полей с печатью данных о найденных объектах», «Редактирование объекта», «Удаление объекта из контейнера», «Сохранение данных всех объектов в файле», «Чтение данных объектов из файла», «Сортировка объектов контейнера по выбранному полю для list», «Печать списка объектов». Предусмотреть обработку исключений, возможные исключения определить самостоятельно. В функции main должен быть главный поток, который создает консольное меню для выбора режима, режимы, требующие взаимодействия с пользователем (ввод нового объекта, редактирование объекта, поиск и печатью, печать списка объектов и т.п.) выполняются в этом главном потоке. Режимы, не требующие взаимодействия с пользователем (удаление, сохранение в файле и чтение из файла), выполняются в отдельном потоке, созданном в главном, при этом обеспечить синхронизацию при доступе к данным объектов.

Тема задания	Контейнер		
	Список list	Неупорядоченное множество unordered_set	Упорядоченное множество set
	Номера вариантов		
Класс сотрудников и класс структурных подразделений (отделов и др.) для отдела кадров организации.	3	10	17

Программа:

Файл “worker.h”:

```
#pragma once
#include<iostream>
#include<fstream>
#include<list>
#include<algorithm>
#include<string>

class Worker {
    std::string name;
    std::string post;
    int age;
    int salary;
public:
    Worker();
    Worker(std::string _name, std::string _post, int _ages, int _salary);
    ~Worker() = default;
    Worker(const Worker& wk);
    Worker(Worker&& wk) noexcept;
    Worker& operator=(const Worker& wk);
    std::string get_name() const;
```

```

std::string get_post() const;
int get_age() const;
int get_salary() const;
void edit_name(std::string _name);
void edit_post(std::string _post);
void edit_age(int _age);
void edit_salary(int _salary);
void init();
friend std::ostream& operator<<(std::ostream& out, const Worker& ob);
friend bool operator==(const Worker& wk_1, const Worker& wk_2);
friend bool operator!=(const Worker& wk_1, const Worker& wk_2);
friend std::istream& operator>>(std::istream& in, Worker& ob);
};

```

Файл “department.h”:

```

#pragma once
#include<iostream>
#include<fstream>
#include<list>
#include<algorithm>
#include<string>
#include"worker.h"

class Department {
    std::string title;
    std::string director_name;
    size_t amount;
    std::list<Worker> dep;
public:
    Department();
    Department(std::string _title, std::string _director_name, size_t _amount,
Worker* _dep);
    ~Department() = default;
    void Add(const Worker& book);
    std::string get_title();
    std::string get_director_name();
    size_t get_amount();
    std::list<Worker>::iterator operator[] (const size_t i);
    void eraser(size_t i1, size_t i2);
    void searcher_name(const std::string& _name);
    void searcher_post(const std::string& _post);
    void searcher_age(const int _age);
    void searcher_salary(const int _salary);
    void sorter(const std::string& str);
    void dep_printer();
    void edit_title(const std::string& _title);
    void edit_director_name(const std::string& _director_name);
    void cleaner();
    friend std::ostream& operator<<(std::ostream& out, Department& dep);
    friend std::istream& operator>>(std::istream& in, Department& dep);
    friend Department operator+(Department& dep, const Worker& wk);
    friend Department operator+(const Worker& wk, Department& dep);
};

```

Файл “worker.cpp”:

```
#include<iostream>
#include<algorithm>
#include<string>
#include<iterator>
#include"worker.h"

Worker::Worker() {
    name = "";
    post = "";
    age = 0;
    salary = 0;
}

Worker::Worker(std::string _name, std::string _post, int _age, int _salary) {
    name = _name;
    post = _post;
    if (_age < 0) throw std::invalid_argument("INVALID AGE ! VALUE: " +
std::to_string(_age));
    age = _age;
    salary = _salary;
}

Worker::Worker(const Worker& wk) {
    name = wk.name;
    post = wk.post;
    age = wk.age;
    salary = wk.salary;
}

Worker::Worker(Worker&& wk) noexcept {
    std::swap(name, wk.name);
    std::swap(post, wk.post);
    std::swap(age, wk.age);
    std::swap(salary, wk.salary);
}

Worker& Worker::operator=(const Worker& wk) {
    name = wk.name;
    post = wk.post;
    age = wk.age;
    salary = wk.salary;
    return *this;
}

std::string Worker::get_name() const {
    return name;
}

std::string Worker::get_post() const {
    return post;
}

int Worker::get_age() const {
    return age;
}

int Worker::get_salary() const {
    return salary;
}

void Worker::edit_name(std::string _name) {
    name = _name;
}

void Worker::edit_post(std::string _post) {
    post = _post;
}

void Worker::edit_age(int _age) {
    if (_age < 0) throw std::invalid_argument("INVALID AGE ! VALUE: " +
std::to_string(_age));
    age = _age;
}
```

```

}
void Worker::edit_salary(int _salary) {
    salary = _salary;
}
void Worker::init() {
    std::cout << "Enter the name of a worker: ";
    std::getline(std::cin, name, '\n');
    std::cout << "Enter the post of a worker: ";
    std::getline(std::cin, post, '\n');
    std::cout << "Enter the age of a worker: ";
    std::cin >> age;
    if (age < 0) throw std::invalid_argument("INVALID AGE ! VALUE: " +
std::to_string(age));
    std::cout << "Enter the salary of worker: ";
    std::cin >> salary;
}

std::ostream& operator<<(std::ostream& out, const Worker& ob) {
    out << "The name of the worker: " << ob.name << std::endl << "The post of
the worker: " << ob.post << std::endl
        << "The age of the worker: " << ob.age << std::endl << "The salary of
the worker : " << ob.salary << std::endl;
    return out;
}

std::istream& operator>>(std::istream& in, Worker& ob) {
    std::string str;
    getline(in, str, ':');
    getline(in, ob.name, '\n');
    getline(in, str, ':');
    getline(in, ob.post, '\n');
    getline(in, str, ':');
    in >> ob.age;
    if (ob.age < 0) throw std::invalid_argument("INVALID AGE ! VALUE: " +
std::to_string(ob.age));
    getline(in, str, ':');
    in.ignore();
    in >> ob.salary;
    return in;
}

bool operator==(const Worker& wk_1, const Worker& wk_2) {
    if (wk_1.get_name() != wk_2.get_name()) {
        return false;
    }
    if (wk_1.get_post() != wk_2.get_post()) {
        return false;
    }
    if (wk_1.get_age() != wk_2.get_age()) {
        return false;
    }
    if (wk_1.get_salary() != wk_2.get_salary()) {
        return false;
    }
    return true;
}

bool operator!=(const Worker& wk_1, const Worker& wk_2) {
    if (wk_1 == wk_2) {
        return false;
    }
    return true;
}

```

Файл “department.cpp”:

```
#include<iostream>
#include<algorithm>
#include<string>
#include<iterator>
#include"worker.h"
#include"department.h"
#include<iterator>
#include<list>

bool sort_title(Worker& wk_1, Worker& wk_2) {
    return wk_1.get_name() < wk_2.get_name();
}

bool sort_author(Worker& wk_1, Worker& wk_2) {
    return wk_1.get_post() < wk_2.get_post();
}

bool sort_year(Worker& wk_1, Worker& wk_2) {
    return wk_1.get_age() < wk_2.get_age();
}

bool sort_pages(Worker& wk_1, Worker& wk_2) {
    return wk_1.get_salary() < wk_2.get_salary();
}

bool operator<(const Worker& wk_1, const Worker& wk_2) {
    return wk_1.get_age() < wk_2.get_age();
}

Department::Department() {
    amount = 0;
    title = "";
    director_name = "";
}

Department::Department(std::string _title, std::string _director_name, size_t
_amount, Worker* _dep) {
    title = _title;
    director_name = _director_name;
    amount = _amount;
    for (size_t i = 0; i < _amount; ++i) {
        dep.push_back(_dep[i]);
    }
}

void Department::Add(const Worker& wk) {
    dep.push_back(wk);
    amount++;
}

std::string Department::get_title() {
    return title;
}

std::string Department::get_director_name() {
    return director_name;
}

size_t Department::get_amount() {
    return amount;
}

std::list<Worker>::iterator Department::operator[](const size_t i) {
    if (i > dep.size()) throw std::logic_error("INDEX OUT OF RANGE! " +
                                                std::to_string(i));

    auto it = dep.begin();
    std::advance(it, i);
    return it;
}

void Department::eraser(size_t i1, size_t i2) {
    if (i1 >= dep.size()) throw std::logic_error("INDEX OUT OF RANGE! " +
```

```

        std::to_string(i1));
    if (i2 >= dep.size()) throw std::logic_error("INDEX OUT OF RANGE! " +
        std::to_string(i2));
    if (i1 && i2 > i2) throw std::logic_error("FIRST INDEX " +
std::to_string(i1) + " IS LOWER THAN SECOND " +
        std::to_string(i2) + "\n");

    auto it = dep.begin();
    std::advance(it, i1);
    if (i1 == i1) {
        dep.erase(it);
        amount--;
    }
    else {
        auto it_end = dep.begin();
        std::advance(it_end, i2);
        dep.erase(it, it_end);
        amount = amount - (i2 - i1 - 1);
    }
}

void Department::searcher_name(const std::string& _name) {
    size_t count = 0;
    auto it = dep.begin();
    Worker obl;
    while (true) {
        obl = *it;
        if (obl.get_name() == _name) {
            std::cout << obl << std::endl;
            count++;
        }
        it++;
        if (it == dep.end()) {
            break;
        }
    }
    std::cout << "Amount of workers with name " << _name << " is: "
        << count << std::endl;
}

void Department::searcher_post(const std::string& _post) {
    size_t count = 0;
    auto it = dep.begin();
    Worker obl;
    while (true) {
        obl = *it;
        if (obl.get_post() == _post) {
            std::cout << obl << std::endl;
            count++;
        }
        it++;
        if (it == dep.end()) {
            break;
        }
    }
    std::cout << "Amount of workers with post " << _post << " is: "
        << count << std::endl;
}

void Department::searcher_age(const int _age) {
    size_t count = 0;
    auto it = dep.begin();
    Worker obl;
    while (true) {
        obl = *it;
        if (obl.get_age() == _age) {
            std::cout << obl << std::endl;
            count++;
        }
    }
}

```

```

    }
    it++;
    if (it == dep.end()) {
        break;
    }
}
std::cout << "Amount of workers with age  " << _age << " is: "
          << count << std::endl;
}
void Department::searcher_salary(const int _salary) {
    size_t count = 0;
    auto it = dep.begin();
    Worker obl;
    while (true) {
        obl = *it;
        if (obl.get_salary() == _salary) {
            std::cout << obl << std::endl;
            count++;
        }
        it++;
        if (it == dep.end()) {
            break;
        }
    }
    std::cout << "Amount of workers with salary  " << _salary << " is: "
          << count << std::endl;
}
void Department::sorter(const std::string& str) {
    if (str == "sort_name") {
        dep.sort(sort_author);
    }
    else if (str == "sort_post") {
        dep.sort(sort_title);
    }
    else if (str == "sort_age") {
        dep.sort(sort_year);
    }
    else if (str == "sort_salary") {
        dep.sort(sort_pages);
    }
    else throw std::invalid_argument("INVALID SORT CHOISE !!! " + str);
}
void Department::dep_printer() {
    std::cout << "Title: " << title << std::endl
          << "\nDirector name: " << director_name << std::endl
          << "\nAmount: " << amount << std::endl;
}
void Department::edit_title(const std::string& _title) {
    title = _title;
}
void Department::edit_director_name(const std::string& _director_name) {
    director_name = _director_name;
}
void Department::cleaner() {
    title = "";
    director_name = "";
    amount = 0;
    dep.clear();
}

std::ostream& operator<<(std::ostream& out, Department& dep) {
    out << "Title: " << dep.title << "Director name: " << dep.director_name
          << "Amount of workers: " << dep.amount << std::endl;
    for (size_t i = 0; i < dep.amount; ++i) {

```



```

        out << "[" << i + 1 << "]" " << *dep[i] << std::endl;
    }
    return out;
}

std::istream& operator>>(std::istream& in, Department& dep) {
    std::string str;
    getline(in, str, ':');
    getline(in, dep.title, '\n');
    getline(in, str, ':');
    getline(in, dep.director_name, '\n');
    while (!in.eof()) {
        Worker ob;
        in >> ob;
        if (ob.get_name().empty() || ob.get_post().empty()) {
            break;
        }
        dep.Add(ob);
    }
    return in;
}

Department operator+(Department& dep, const Worker& wk) {
    dep.Add(wk);
    return dep;
}

Department operator+(const Worker& wk, Department& dep) {
    dep.Add(wk);
    return dep;
}

```

Файл “main.cpp”:

```

#include<iostream>
#include<fstream>
#include<mutex>
#include<future>
#include<list>
#include<algorithm>
#include<string>
#include<thread>
#include <stdio.h>
#include"worker.h"
#include"department.h"

std::mutex mut_1;
void r_file(Department& dep, const std::string& str) {
    mut_1.lock();
    std::ifstream fin(str);
    if (!fin.is_open()) {
        std::cout << "NOT FOUND SUCH FILE! " << std::endl;
    }
    dep.cleaner();
    fin >> dep;
    fin.close();
    mut_1.unlock();
}

std::mutex mut_2;
void eraser(Department& dep, const size_t number, bool& saved) {
    mut_2.lock();
    if (number != 0) {

```

```

        try {
            dep.eraser(number - 1, number - 1);
            saved = false;
        }
        catch (std::logic_error& error) {
            std::cout << error.what();
        }
    }
    else std::cout << "WRONG VALUE ! ";
    mut_2.unlock();
}

std::mutex mut_3;
void saver(Department& dep, const std::string& str, bool& saved) {
    mut_3.lock();
    try {
        std::ofstream fout(str);
        fout << dep;
        saved = true;
    }
    catch (const std::exception& error) {
        std::cout << error.what() << std::endl;
        std::cout << "NOT SUCH FILE FOUND ! " << std::endl;
    }
    mut_3.unlock();
}

int main() {
    std::string file;
    int choice;
    Department Dep;
    std::cout << "Enter name of department:\n";
    //std::cin.ignore();
    getline(std::cin, file);
    std::ifstream fin(file);
    fin >> Dep;
    fin.close();
    bool saved = true;
    while (true) {
        std::cout << "Choice what you want to do: "
            "\n[0] - Back"
            "\n[1] - Load department from file"
            "\n[2] - Print info of department"
            "\n[3] - Add worker in department"
            "\n[4] - Delete worker from department"
            "\n[5] - Find worker by parametr"
            "\n[6] - Sort department by parametr"
            "\n[7] - Edit department info"
            "\n[8] - Save department info a file"
            "\n[9] - Create new department"
            "\n-> ";

        std::cin >> choice;
        if (choice == 0) {
            if (!saved) {
                int local_choise;
                std::cout << "There are not saved changes! "
                    "\n ENTER any button to return and 0 to confirm
an exit";

                std::cin >> local_choise;
                if (local_choise == 0) {
                    break;
                }
                else break;
            }
        }
    }
}

```

```

    }
    switch (choice)
    {
        case 0: {
            std::cout << "\n\n----- SUCCESSFULLY END ! -----
            -----\n\n";
            system("pause");
            return 0;
        }
        case 1: {
            std::string str;
            std::cout << "Enter the path to file: ";
            std::cin.ignore();
            std::getline(std::cin, str, '\n');
            std::thread thread_1(r_file, std::ref(Dep), std::ref(str));
            thread_1.join();
            saved = false;
            break;
        }
        case 2: {
            std::cout << Dep;
            break;
        }
        case 3: {
            Worker wk;
            try {
                std::cin.ignore();
                wk.init();
                Dep.Add(wk);
                saved = false;
            }
            catch (const std::exception& error) {
                std::cout << error.what() << std::endl;
            }
            break;
        }
        case 4: {
            size_t number;
            std::cout << "Enter the number of worker, which you want to
delete: ";

            std::cin >> number;
            std::thread thread_buf(eraser, std::ref(Dep), number,
std::ref(saved));
            thread_buf.join();
            break;
        }
        case 5: {
            int local_choice;
            std::cout << "Chose parametr of searching: "
                "\n[0] - back"
                "\n[1] - Search by name"
                "\n[2] - Search by post"
                "\n[3] - Search by age"
                "\n[4] - Search by salary"
                "\n-> ";
            std::cin >> local_choice;
            switch (local_choice)
            {
                case 0: {
                    break;
                }
                case 1: {
                    std::string name;
                    std::cout << "Enter name of the worker: ";

```

```

        std::cin.ignore();
        std::getline(std::cin, name, '\n');
        Dep.searcher_name(name);
        break;
    }
    case 2: {
        std::string post;
        std::cout << "Enter post of the worker: ";
        std::cin.ignore();
        std::getline(std::cin, post, '\n');
        Dep.searcher_post(post);
        break;
    }
    case 3: {
        int age;
        std::cout << "Enter age of the worker: ";
        std::cin >> age;
        try {
            Dep.searcher_age(age);
        }
        catch (std::invalid_argument& error) {
            std::cout << error.what();
        }
        break;
    }
    case 4: {
        int salary;
        std::cout << "Enter salary of the worker: ";
        std::cin >> salary;
        Dep.searcher_salary(salary);
        break;
    }
    default:
        std::cout << "ERROR ! Wrong comand !" << std::endl;
    }
    break;
}
case 6: {
    int local_choice;
    std::cout << "Choice type of sorting: "
                "\n[0] - back"
                "\n[1] - Sort by name"
                "\n[2] - Sort by post"
                "\n[3] - Sort by age"
                "\n[4] - Sort by salary"
                "\n-> ";
    std::cin >> local_choice;
    switch (local_choice)
    {
        case 0: {
            break;
        }
        case 1: {
            Dep.sorter("sort_name");
            std::cout << "Library after sorting by name: \n" <<
Dep;

            saved = false;
            break;
        }
        case 2: {
            Dep.sorter("sort_post");
            std::cout << "Library after sorting by post: \n" <<
Dep;

            saved = false;

```

```

        break;
    }
    case 3: {
        Dep.sorter("sort_age");
        std::cout << "Library after sorting by age: \n" <<

Dep;

        saved = false;
        break;
    }
    case 4: {
        Dep.sorter("sort_salary");
        std::cout << "Library after sorting by salary: \n" <<

Dep;

        saved = false;
        break;
    }
    default:
        std::cout << "ERROR ! Wrong command !" << std::endl;
    }
    break;
}
case 7: {
    int local_choice;
    std::cout << "Choice parametr of department which you want to

edit: "

        "\n[0] - back"
        "\n[1] - Edit title"
        "\n[2] - Edit director name"
        "\n[3] - Edit worker info"
        "\n-> ";
    std::cin >> local_choice;
    switch (local_choice)
    {
        case 0: {
            break;
        }
        case 1: {
            std::string title;
            std::cout << "Enter new title: ";
            std::cin.ignore();
            std::getline(std::cin, title, '\n');
            Dep.edit_title(title);
            saved = false;
            Dep.dep_printer();
            break;
        }
        case 2: {
            std::string director_name;
            std::cout << "Enter new director name: ";
            std::cin.ignore();
            std::getline(std::cin, director_name, '\n');
            Dep.edit_director_name(director_name);
            saved = false;
            Dep.dep_printer();
            break;
        }
        case 3: {
            size_t number;
            std::cout << "Enter the number of the worker: ";
            std::cin >> number;
            try {
                int local_choice_2;
                std::cout << "Choice parametr which you want to

edit: "

```

```

        "\n[0] - back"
        "\n[1] - Name of the worker"
        "\n[2] - Post of the worker"
        "\n[3] - Age of the worker"
        "\n[4] - Salary of the worker"
        "\n-> ";
std::cin >> local_choice_2;
switch (local_choice_2)
{
    case 0: {
        break;
    }
    case 1: {
        std::string name;
        std::cout << "Enter new name: ";
        std::cin.ignore();
        std::getline(std::cin, name, '\n');
        Dep[number - 1]->edit_name(name);
        saved = false;
        break;
    }
    case 2: {
        std::string post;
        std::cout << "Enter new post";
        std::cin.ignore();
        std::getline(std::cin, post, '\n');
        Dep[number - 1]->edit_post(post);
        saved = false;
        break;
    }
    case 3: {
        int age;
        std::cout << "Enter new age: ";
        std::cin >> age;
        try {
            Dep[number - 1]->edit_age(age);
            saved = false;
        }
        catch (std::invalid_argument& error) {
            std::cout << error.what();
        }
        break;
    }
    case 4: {
        int salary;
        std::cout << "Enter new salary: ";
        std::cin >> salary;
        Dep[number - 1]->edit_salary(salary);
        saved = false;
        break;
    }
    default:
        std::cout << "ERROR ! Wrong command !" <<
std::endl;
    }
}
catch (std::logic_error& error) {
    std::cout << error.what();
}
break;
}
default:
    std::cout << "ERROR ! Wrong command !" << std::endl;
}

```

```

        break;
    }
    case 8: {
        int local_choice;
        std::cout << "Save changes ?"
                    "\n[0] - back"
                    "\n[1] - YES"
                    "\n[2] - NO"
                    "\n-> ";
        std::cin >> local_choice;
        switch (local_choice)
        {
            case 0: {
                break;
            }
            case 1: {
                if (saved) {
                    std::cout << "Your data has already saved !";
                    break;
                }
                std::ofstream fout(file);
                fout << Dep;
                saved = true;
                break;
            }
            case 2: {
                std::string file;
                std::cout << "Enter full path to file: ";
                std::cin.ignore();
                std::getline(std::cin, file, '\n');
                std::thread thread_3(saver, std::ref(Dep),
std::ref(file), std::ref(saved));
                thread_3.join();
                break;
            }
            default:
                std::cout << "ERROR ! Wrong command !" << std::endl;
        }
        break;
    }
    case 9: {
        if (!saved) {
            std::cout << "There are some not saved changes ! "
                        "\nCreating new library will delete them!"
                        "\n[0] - back"
                        "\n[1] - continue"
                        "\n-> ";

            int local_choice;
            std::cin >> local_choice;
            switch (local_choice) {
                case 0: {
                    break;
                }
                case 1: {
                    Dep.cleaner();
                    saved = false;
                    break;
                }
                default:
                    std::cout << "ERROR !Wrong command !" <<
std::endl;
            }
        }
        else Dep.cleaner();
    }
}

```

```

        break;
    }
    default:
        std::cout << "ERROR ! Wrong command !" << std::endl;
    }
}
return 0;
}

```

Пример работы программы:

- 1) Читаем заготовленный файл и выводим его данные:

```

Choice what you want to do:
[0] - Back
[1] - Load department from file
[2] - Print info of department
[3] - Add worker in department
[4] - Delete worker from department
[5] - Find worker by parametr
[6] - Sort department by parametr
[7] - Edit department info
[8] - Save department info a file
[9] - Create new department
-> 1
Enter the path to file: C:\Users\goza-\source\repos\DZ3\first.txt
Choice what you want to do:
[0] - Back
[1] - Load department from file
[2] - Print info of department
[3] - Add worker in department
[4] - Delete worker from department
[5] - Find worker by parametr
[6] - Sort department by parametr
[7] - Edit department info
[8] - Save department info a file
[9] - Create new department
-> 2
Title:  first Director name: John Amount of workers: 3
[1] The name of the worker: Paul
The post of the worker: intern
The age of the worker: 19
The salary of the worker : 5000

[2] The name of the worker: Bob
The post of the worker: junior
The age of the worker: 28
The salary of the worker : 0

[3] The name of the worker: Kay
The post of the worker: junior
The age of the worker: 30
The salary of the worker : 5000

```


2) Добавляем сотрудника в department:

```
Choice what you want to do:
[0] - Back
[1] - Load department from file
[2] - Print info of department
[3] - Add worker in department
[4] - Delete worker from department
[5] - Find worker by parametr
[6] - Sort department by parametr
[7] - Edit department info
[8] - Save department info a file
[9] - Create new department
-> 3
Enter the name of a worker: Joe
Enter the post of a worker: middle
Enter the age of a worker: 38
Enter the salary of worker: 850000
```

3) Ищем сотрудника по параметру (например, по возрасту):

```
Chose parametr of searching:
[0] - back
[1] - Search by name
[2] - Search by post
[3] - Search by age
[4] - Search by salary
-> 3
Enter age of the worker: 19
The name of the worker: Paul
The post of the worker: intern
The age of the worker: 19
The salary of the worker : 25000

Amount of workers with age 19 is: 1
```

4) Сортируем сотрудников (например, по зарплате):

```
Choice what you want to do:
[0] - Back
[1] - Load department from file
[2] - Print info of department
[3] - Add worker in department
[4] - Delete worker from department
[5] - Find worker by parametr
[6] - Sort department by parametr
[7] - Edit department info
[8] - Save department info a file
[9] - Create new department
-> 6
Choice type of sorting:
[0] - back
[1] - Sort by name
[2] - Sort by post
[3] - Sort by age
[4] - Sort by salary
-> 4
Library after sorting by salary:
Title: first Director name: John Amount of workers: 4
[1] The name of the worker: Paul
The post of the worker: intern
The age of the worker: 19
The salary of the worker : 25000

[2] The name of the worker: Kay
The post of the worker: junior
The age of the worker: 30
The salary of the worker : 45000

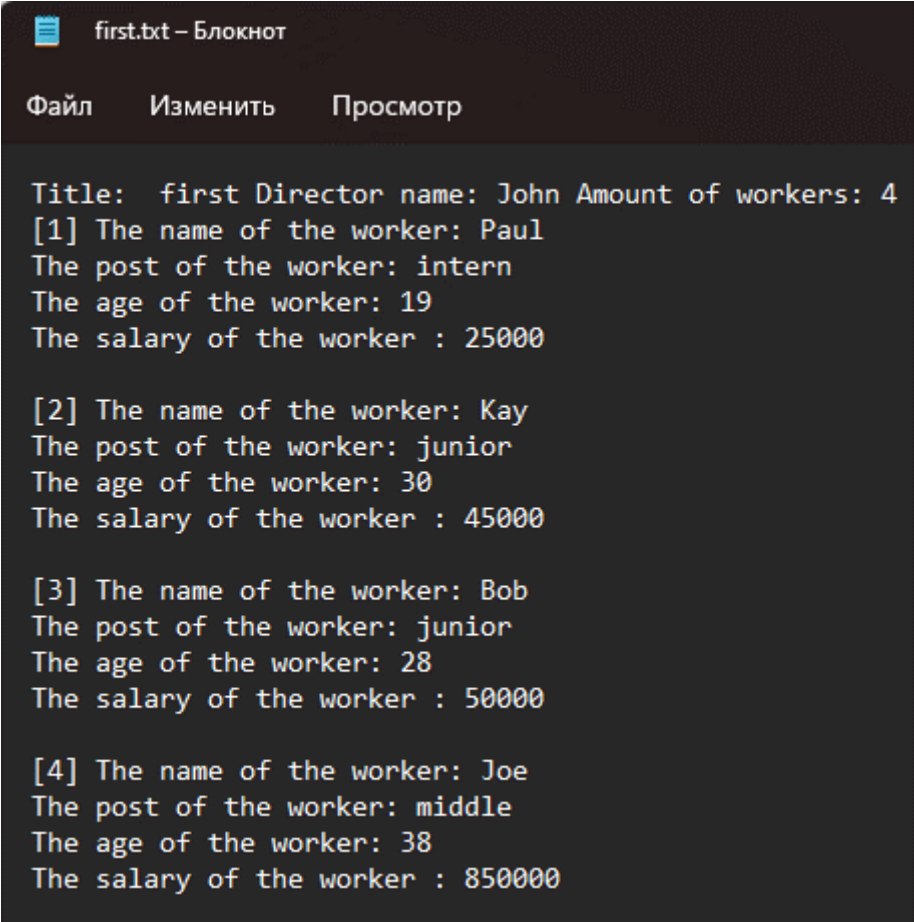
[3] The name of the worker: Bob
The post of the worker: junior
The age of the worker: 28
The salary of the worker : 50000

[4] The name of the worker: Joe
The post of the worker: middle
The age of the worker: 38
The salary of the worker : 850000
```

5) Сохраняем изменения в файл:

```
Choice what you want to do:
[0] - Back
[1] - Load department from file
[2] - Print info of department
[3] - Add worker in department
[4] - Delete worker from department
[5] - Find worker by parametr
[6] - Sort department by parametr
[7] - Edit department info
[8] - Save department info a file
[9] - Create new department
-> 8
Save changes ?
[0] - back
[1] - YES
[2] - NO
-> 1
```

6) Файл, в который сохранились изменения:



first.txt – Блокнот

Файл Изменить Просмотр

Title: first Director name: John Amount of workers: 4

[1] The name of the worker: Paul
The post of the worker: intern
The age of the worker: 19
The salary of the worker : 25000

[2] The name of the worker: Kay
The post of the worker: junior
The age of the worker: 30
The salary of the worker : 45000

[3] The name of the worker: Bob
The post of the worker: junior
The age of the worker: 28
The salary of the worker : 50000

[4] The name of the worker: Joe
The post of the worker: middle
The age of the worker: 38
The salary of the worker : 850000