|  |  |
| --- | --- |
|  | **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 № 4

ПО КУРСУ

«Алгоритмические языки»

на тему «**Использование объектов своих классов в последовательных контейнерах библиотеки STL**»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент | ИУ8-23 |  |  |  | Д.С. Афанасьев |
|  | (Группа) |  |  |  | (И. О. Фамилия) |
|  |  |  |  |  |  |
| Преподаватель: |  |  |  |  | М. В. Малахов |
|  |  |  |  |  | (И.О. Фамилия) |

2024

Введение

Условия для 4 варианта

В приложении организовать контейнер объектов своего класса (использовать шаблоны std::list, std::vector или std::deque в зависимости от варианта, элементы контейнера - объекты класса, **не указатели!!!!**). Варианты заданий заданы в ячейках таблицы 1. Параметры приложений указаны в заголовках строк и столбцов таблицы 1.

Класс должен иметь необходимые конструкторы, конструктор копирования и перемещения при необходимости (обосновать отсутствие или наличие необходимости), перегруженные операции присваивания с копированием и перемещением при необходимости (обосновать отсутствие или наличие необходимости), перегруженную операцию вставки в поток <<.

Обеспечить копирование одного контейнера в другой с помощью алгоритма std::copy. А также сортировку объектов в исходном контейнере, для шаблона list при сортировке использовать метод list::sort без параметров, для шаблона vector или deque при сортировке использовать алгоритм std::sort с двумя параметрами: итератор на начало и итератор на конец контейнера.

Исходные данные прочитать из текстового файла input.txt. Вывести в выходной файл output.txt исходный контейнер, контейнер после сортировки, использовать при этом перегруженную операцию вставки в поток, также вывести в выходной файл контейнер, в который скопирован исходный контейнер.

|  |  |  |
| --- | --- | --- |
| **Параметры приложений** | | **Исходный контейнер list, копируем в vector** |
| Объект- сотрудник (поля: ФИО, дата приема на работу, должность, базовый оклад) | Сортировка по окладу | 4 |
|  |  |

Основная часть

Исходный текст программы:

Employee.h:

#include <iostream>

#include <string>

#ifndef EMPLOYEE\_H

#define EMPLOYEE\_H

class Employee {

protected:

  std::string fio;

  std::string hireDate;

  std::string position;

  int salary;

public:

  Employee();

  Employee(std::string, std::string, std::string, int);

  Employee(const Employee&);

  std::string getFio() const;

  std::string getHireDate() const;

  std::string getPosition() const;

  int getSalary() const;

*// .Employee(Employee&);*

  void updateSalary(int);

  Employee& operator = (const Employee&);

};

std::ostream& operator << (std::ostream&, const Employee&);

#endif

Employee.cpp:

#include "Employee.h"

#include <iostream>

#include <string>

Employee::Employee() {

  fio = hireDate = position = "undefined";

  salary = 0;

}

Employee::Employee(std::string fio, std::string hireDate, std::string position, int salary) {

*this*->fio = fio;

*this*->hireDate = hireDate;

*this*->position = position;

*this*->salary = salary;

}

Employee::Employee(const Employee& employee) {

  fio = employee.fio;

  hireDate = employee.hireDate;

  position = employee.position;

  salary = employee.salary;

}

std::string Employee::getFio() const {

  return fio;

}

std::string Employee::getHireDate() const {

  return hireDate;

}

std::string Employee::getPosition() const {

  return position;

}

int Employee::getSalary() const {

  return salary;

}

void Employee::updateSalary(int salary) {

*this*->salary = salary;

}

Employee& Employee::operator = (const Employee& employee) {

*this*->fio = employee.getFio();

*this*->hireDate = employee.getHireDate();

*this*->position = employee.getPosition();

*this*->salary = employee.getSalary();

  return \**this*;

}

std::ostream& operator << (std::ostream& out, const Employee& employee) {

  return out << "\tFIO: " << employee.getFio() << std::endl

      << "\thire date: " << employee.getHireDate() << std::endl

      << "\tposition: " << employee.getPosition() << std::endl

      << "\tsalary: " << employee.getSalary() << std::endl;

}

lab4.cpp:

#include "Employee.h"

#include <algorithm>

#include <fstream>

#include <iostream>

#include <list>

#include <string>

#include <vector>

*// g++ source/lab4/lab4.cpp source/lab4/Employee.cpp -o build/lab4*

std::string tmp;

int main() {

  std::list<Employee> employees\_list;

  std::vector<Employee> employees\_vec;

  std::ifstream fin("./input/input\_lab4.txt");

  if (!fin.is\_open()) {

    std::cerr << "Error: Couldn't open input file" << std::endl;

    return 1;

  }

  int employees\_count;

  getline(fin, tmp);

  try {

    employees\_count = std::stoi(tmp);

  } catch (const std::invalid\_argument &e) {

    std::cerr << "Error: Invalid employees' count" << std::endl;

    std::cerr << e.what() << std::endl;

    fin.close();

    return 1;

  }

  std::string fio;

  std::string hireDate;

  std::string position;

  int salary;

  for (int i = 0; i < employees\_count; ++i) {

    getline(fin, fio);

    getline(fin, hireDate);

    getline(fin, position);

    getline(fin, tmp);

    try {

      salary = std::stoi(tmp);

    } catch (const std::invalid\_argument &e) {

      std::cerr << "Error: Invalid salary" << std::endl;

      std::cerr << e.what() << std::endl;

      fin.close();

      return 1;

    }

    employees\_list.push\_back(Employee(fio, hireDate, position, salary));

  }

  fin.close();

  employees\_vec.resize(employees\_count);

  std::copy(employees\_list.begin(), employees\_list.end(), employees\_vec.begin());

  employees\_list.sort([](Employee a, Employee b) {

    return a.getSalary() > b.getSalary();

  });

  std::ofstream out\_list, out\_vec;

  out\_vec.open("./output/out\_vec\_lab4.txt");

  if (!out\_vec.is\_open()) {

    std::cerr << "Error: Cannot open vector output file\n";

    return 1;

  }

  int i = 1;

  for (Employee& employee : employees\_vec) {

    out\_vec << "\t\t==========Employee " << i++ << " ==========\n";

    out\_vec << employee;

  }

  out\_vec.close();

  out\_list.open("./output/out\_list\_lab4.txt");

  if (!out\_list.is\_open()) {

    std::cerr << "Error: Cannot open list output file\n";

    return 1;

  }

  i = 1;

  for (Employee& employee : employees\_list) {

    out\_list << "\t\t==========Employee " << i++ << " ==========\n";

    out\_list << employee;

  }

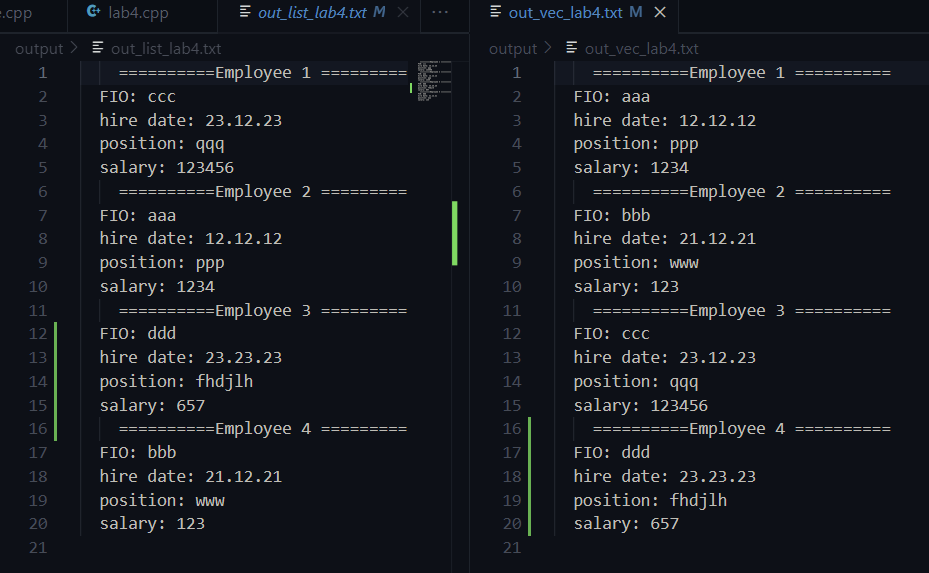
  out\_list.close();

  return 0;

}

Снимки выполнения работы программы





Заключение

Задачи лабораторной работы были решены, результаты проверены. Изучены на практике основные понятия объектно-ориентированного программирования языка Си++ – классов и объектов и использования их вместе с контейнерами из библиотеки STL, и приобретены навыки разработки программ на языке Си++ с использованием объектно- ориентированных средств.