# Министерство образования и науки Российской Федерации

## Федеральное государственное бюджетное образовательное учреждение высшего образования

## «Новосибирский государственный технический университет»



## Кафедра теоретической и прикладной информатики

### Лабораторная работа № 2 по дисциплине «Объектно-ориентированные технологии разработки программного обеспечения»

**Последовательные контейнеры STL**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Факультет: | ПМИ |  |
| Группа: | ПМИМ-01 |
| Студенты: | Ершов П.К., Малышкина Е.Д.  Слободчикова А.Э. |
| Вариант: | 4 |
| Преподаватель: | Лисицин Д. В. |

Новосибирск

2021

1. **Цель работы**

Изучить последовательные контейнеры и способы работы с ними.

1. **ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ**
2. Изучить элементы стандартной библиотеки шаблонов С++: последовательные контейнеры, адаптеры контейнеров, функциональные объекты, функциональные адаптеры, лямбда-выражения.
3. Переработать программу из работы №1 так, чтобы объекты разработанного класса хранились в одном из следующих контейнеров STL: список (list), односвязный список (forward\_list). Использовать функции-члены контейнера для реализации следующих возможностей:
   1. Добавление объектов в контейнер.
   2. Поиск объектов в контейнере по различным критериям.
   3. Модификация объектов в контейнере (возможно, в соответствии с некоторыми критериями).
   4. Удаление объектов из контейнера по различным критериям.
   5. Вывод всех объектов контейнера, отсортированных по различным критериям, в файл и/или на экран.
4. При формировании критериев поиска, удаления, сортировки использовать функциональные объекты (пользовательские и/или стандартные) и/или лямбда-выражения. Атрибуты класса, хранящиеся как строки, но по смыслу являющиеся числами, должны участвовать в сравнениях как числа.
5. Оформить отчет. Отчет должен содержать постановку задачи, алгоритм, описание и текст разработанной программы, результаты тестирования (со скриншотами) и выводы.
6. Защитить работу, ответив на вопросы преподавателя.
7. **Вариант задания**

Класс с именем WORKER, содержащий следующие атрибуты:

– фамилия работника;

– имя работника;

– отчество работника;

– название занимаемой должности;

– год поступления на работу.

Находить работников, которые имеют

– стаж работы в организации, превышающий заданное значение;

– заданную должность

1. **Описание алгоритма**

Worker(); - конструктор экземпляра класса по умолчанию

Worker(string s\_name, string name, string patronymic, string function, int day, int month, int year); - конструктор экземпляра класса с заданными параметрами

int checkDate(int day, int month, int year); - функция проверки формата даты

void set\_Date(int day, int month, int year); - функция установки даты

void setS\_name(string s\_name); - функция установки фамилии

void setName(string name); - функция установки имени

void setPatr(string patronymic); - функция установки отчества

void setFunc(string function); - функция установки должности

template <typename charT, typename traits> friend basic\_ostream<charT, traits>&

operator << (basic\_ostream<charT, traits>& out, const Worker& c); - перегрузка функции потока вывода

friend istream& operator>>(istream& in, Worker& c); - перегрузка функции потока ввода

string getS\_name() - функция получения фамилии

string getName() - функция получения имени

string getPatr() - функция получения отчества

string getFunc() - функция получения должности

int getDay() - функция получения дня

int getMonth() - функция получения месяца

int getYear() - функция получения года

void file\_save(vector<Worker> workers, string filename) – функция получения записей из файла

void output\_all(vector<Worker> workers, int format) – функция сохранения записей в файл

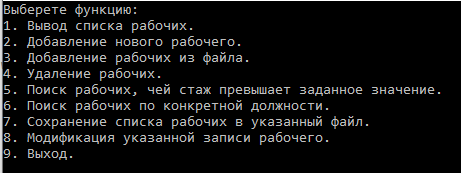
vector<Worker> search(vector<Worker> workers, string function) – функция поиска рабочих по должности

vector<Worker> search(vector<Worker> workers, int year) – функция поиска рабочих по стажу

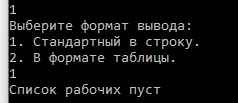
void mod(vector<Worker> &workers, Worker buf, int num) – функция модификации записи рабочего

1. **Примеры работы программы**

**Рабочее меню программы**



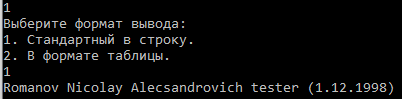
**Результат попытки вывести список рабочих, когда он ещё не загружен из файла (так же можно выбрать формат вывода, реализуемый пользовательскими флагами формата)**



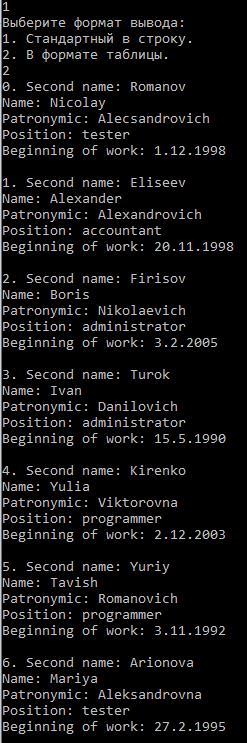
**Добавление новой записи в список**



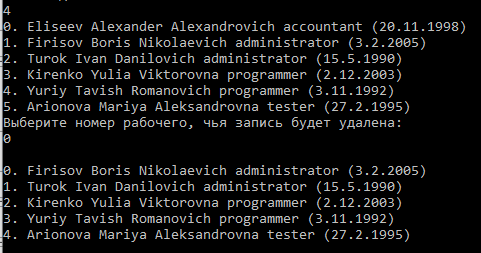
Демонстрация, что запись уже в списке



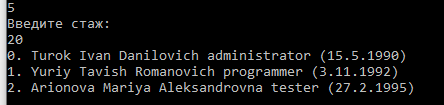
**Вывод списка рабочих после получения списка из файла (вывод форматированный)**

****

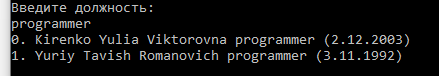
**Демонстрация удаления выбранной записи**

****

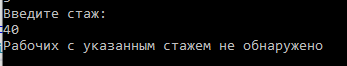
**Поиск рабочих, чей стаж превышает заданное число лет**

****

**Поиск сотрудников по их должности**

****

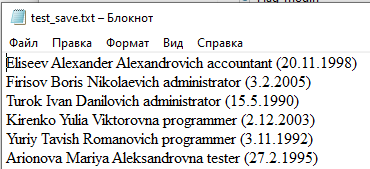
**Демонстрация результатов поиска для случаев, когда не найден ни один сотрудник**

****

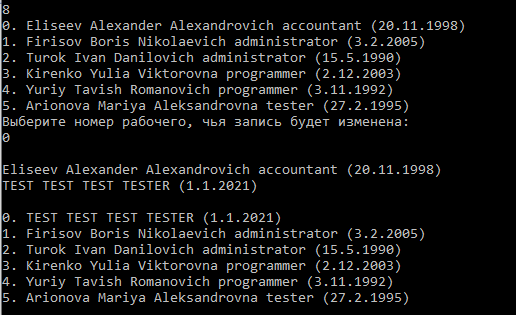
****

**Сохранение списка рабочих в указанный файл**

****

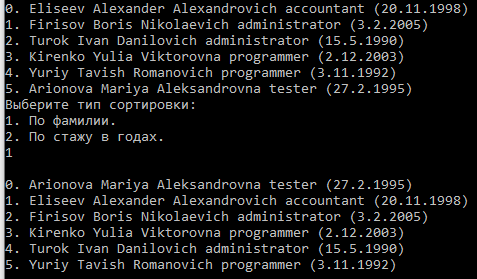
****

**Модификация выбранной записи**

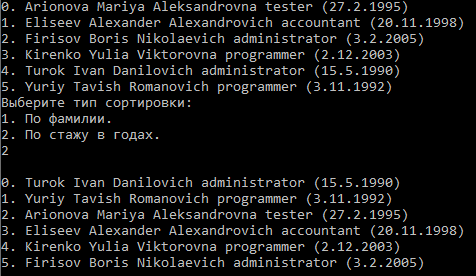
****

**Сортировка рабочих**

**По фамилии**

****

**По стажу**

****

1. **Текст программы**

Файл Worker.h

#ifndef WORKER

#define WORKER

#pragma once

#include <iostream>

#include <iomanip>

#include <sstream>

#include <string>

#include <ctime>

using namespace std;

namespace Workers

{

const int spec\_index = ios\_base::xalloc();

template <typename charT, typename traits>inline basic\_ostream <charT, traits>&

tab\_out(basic\_ostream<charT, traits>& out)

{

out.iword(spec\_index) = true;

return out;

}

template <typename charT, typename traits> inline basic\_ostream<charT, traits>&

stand\_out(basic\_ostream<charT, traits>& out)

{

out.iword(spec\_index) = false;

return out;

}

class Worker

{

private:

string s\_name = "",

name = "",

patronymic = "",

function = "";

int day,

month,

year;

public:

Worker();

Worker(string s\_name, string name, string patronymic, string function, int day, int month, int year);

int checkDate(int day, int month, int year);

void set\_Date(int day, int month, int year);

void setS\_name(string s\_name);

void setName(string name);

void setPatr(string patronymic);

void setFunc(string function);

template <typename charT, typename traits> friend basic\_ostream<charT, traits>&

operator << (basic\_ostream<charT, traits>& out, const Worker& c);

friend istream& operator>>(istream& in, Worker& c);

string getS\_name()

{

return this->s\_name;

}

string getName()

{

return this->name;

}

string getPatr()

{

return this->patronymic;

}

string getFunc()

{

return this->function;

}

int getDay()

{

return this->day;

}

int getMonth()

{

return this->month;

}

int getYear()

{

return this->year;

}

};

template <typename charT, typename traits>

inline basic\_ostream<charT, traits>& operator << (basic\_ostream<charT, traits>& out, const Worker& c)

{

basic\_ostringstream <charT, traits> s;

s.copyfmt(out);

s.width(0);

if (s.iword(spec\_index)) {

s << "Second name: " << c.s\_name << endl;

s << "Name: " << c.name << endl;

s << "Patronymic: " << c.patronymic << endl;

s << "Position: " << c.function << endl;

s << "Beginning of work: " << c.day << "." << c.month << "." << c.year << endl;

}

else

s << c.s\_name << " " << c.name << " " << c.patronymic << " " << c.function << " (" << c.day << "." << c.month << "." << c.year << ")";

out << s.str();

return out;

}

}

#endif

Файл Worker.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "Worker.h"

using namespace Workers;

Worker::Worker()

{

setS\_name(string());

setName(string());

setPatr(string());

this->day = 0;

this->month = 0;

this->year = 0;

}

Worker::Worker(string s\_name, string name, string patronymic, string function, int day, int month, int year)

{

setS\_name(s\_name);

setName(name);

setPatr(patronymic);

setFunc(function);

set\_Date(day, month, year);

}

int Worker::checkDate(int day, int month, int year)

{

struct tm\* date;

time\_t t = time(NULL);

date = gmtime(&t);

if (day > 0)

{

if ((month == 1 || month == 3 || month == 5 || month == 7 ||

month == 8 || month == 10 || month == 12) && day > 31)

return 1;

if ((month == 4 || month == 6 || month == 9 || month == 11) && day > 30)

return 1;

if (month == 2 && year % 4 == 0 && day > 29)

return 1;

if (month == 2 && year % 4 != 0 && day > 28)

return 1;

}

else

return 1;

if (month < 0 || month > 12)

return 2;

if (year < 0 || year > date->tm\_year + 1900)

return 3;

return 0;

}

void Worker::set\_Date(int day, int month, int year)

{

if (checkDate(day, month, year) == 0)

{

this->day = day;

this->month = month;

this->year = year;

}

}

void Worker::setS\_name(string s\_name)

{

this->s\_name = s\_name;

}

void Worker::setName(string name)

{

this->name = name;

}

void Worker::setPatr(string patronymic)

{

this->patronymic = patronymic;

}

void Worker::setFunc(string function)

{

this->function = function;

}

namespace Workers

{

istream& operator>>(istream& in, Worker& input)

{

string s\_name, name, part, func;

int day, month, year;

char open, close, point1, point2;

bool flag = true;

in >> s\_name >> name >> part >> func >> open >> day >> point1 >> month >> point2 >> year >> close;

Worker buf("0", "0", "0", "0", day, month, year);

if (open != '(' || close != ')' || point1 != '.' || point2 != '.')

{

flag = false;

}

if (flag && buf.checkDate(day, month, year) == 0)

{

input.setS\_name(s\_name);

input.setName(name);

input.setPatr(part);

input.setFunc(func);

input.set\_Date(day, month, year);

}

else

{

in.clear(ios::failbit);

return in;

}

return in;

}

}

Файл WORKER\_Class.cpp

#include <iostream>

#include "Worker.h"

#include <fstream>

#include <vector>

#include <list>

using namespace std;

using namespace Workers;

void file\_load(list<Worker>& workers, string filename)

{

Worker wr;

ifstream f(filename);

if (!f.is\_open())

{

cout << "Error! No file with name " << filename << endl;

return;

}

while (!f.eof())

{

f >> wr;

if (f.fail())

{

f.clear(ios::eofbit);

break;

}

workers.push\_back(wr);

}

f.close();

};

void file\_save(list<Worker> workers, string filename)

{

Worker wr;

ofstream f(filename);

if (!f.is\_open())

{

cout << "Error! No file with name " << filename << endl;

return;

}

for (auto iter = workers.begin(); iter != workers.end(); iter++)

{

f << \*iter << endl;

}

f.close();

};

void output\_all(list<Worker> workers, int format)

{

int i = 0;

for (auto iter = workers.begin(); iter != workers.end(); iter++)

{

if (format == 1)

cout << stand\_out << i << ". " << \*iter << endl;

else

cout << tab\_out << i << ". " << \*iter << endl;

i++;

}

}

list<Worker> search(list<Worker> workers, string function)

{

list<Worker> out;

list<Worker>::iterator iter;

iter = find\_if(workers.begin(),

workers.end(),

[&](Worker w)

{

return w.getFunc() == function;

});

while (iter != workers.end())

{

out.push\_back(\*iter);

iter++;

iter = find\_if(iter, workers.end(), [&](Worker w)

{

return w.getFunc() == function;

});

}

return out;

}

list<Worker> search(list<Worker> workers, int year)

{

struct tm\* date;

time\_t t = time(NULL);

date = gmtime(&t);

int tt = date->tm\_year + 1900;

list<Worker> out;

list<Worker>::iterator iter;

iter = find\_if(workers.begin(),

workers.end(),

[&](Worker w)

{

return (tt - w.getYear()) > year;

});

while (iter != workers.end())

{

out.push\_back(\*iter);

iter++;

iter = find\_if(workers.begin(),

workers.end(),

[&](Worker w)

{

return (tt - w.getYear()) > year;

});

}

return out;

};

void sort(list<Worker>& workers, int num)

{

if (num == 1)

workers.sort([](Worker& a, Worker& b) { return a.getS\_name() < b.getS\_name(); });

if (num == 2)

workers.sort([](Worker& a, Worker& b) { return a.getYear() < b.getYear(); });

}

void mod(list<Worker>& workers, Worker buf, int num)

{

auto w\_i = workers.begin();

advance(w\_i, num);

w\_i->setS\_name(buf.getS\_name());

w\_i->setName(buf.getName());

w\_i->setPatr(buf.getPatr());

w\_i->setFunc(buf.getFunc());

w\_i->set\_Date(buf.getDay(), buf.getMonth(), buf.getYear());

};

int main()

{

setlocale(LC\_ALL, "Russian");

bool exit = false;

int option;

list<Worker> workers;

while (!exit)

{

cout << "Выберете функцию:" << endl;

cout << "1. Вывод списка рабочих." << endl;

cout << "2. Добавление нового рабочего." << endl;

cout << "3. Добавление рабочих из файла." << endl;

cout << "4. Удаление рабочих." << endl;

cout << "5. Поиск рабочих, чей стаж превышает заданное значение." << endl;

cout << "6. Поиск рабочих по конкретной должности." << endl;

cout << "7. Сохранение списка рабочих в указанный файл." << endl;

cout << "8. Модификация указанной записи рабочего." << endl;

cout << "9. Сортировка списка." << endl;

cout << "10. Выход." << endl;

cin >> option;

while (cin.fail())

{

cin.clear();

cout << "Не верное значение!" << endl;

cin.ignore();

cin >> option;

}

switch (option)

{

case 1:

{

int format;

cout << "Выберите формат вывода:" << endl;

cout << "1. Стандартный в строку." << endl;

cout << "2. В формате таблицы." << endl;

cin >> format;

if (!workers.empty())

output\_all(workers, format);

else

cout << "Список рабочих пуст" << endl;

cout << endl;

break;

}

case 2:

{

Worker buf;

cout << "Введите данные нового рабочего в формате: Фамилия Имя Отчество Должность (0.0.0000)" << endl;

cin >> buf;

/\*if (cin.fail())

cout << "Ошибка. Неверный формат ввода." << endl;

else\*/

workers.push\_back(buf);

cout << endl;

break;

}

case 3:

{

file\_load(workers, "data.txt");

cout << endl;

break;

}

case 4:

{

if (!workers.empty())

output\_all(workers, 1);

else

{

cout << "Список рабочих пуст" << endl;

break;

}

int num;

cout << "Выберите номер рабочего, чья запись будет удалена:" << endl;

cin >> num;

auto w\_i = workers.begin();

advance(w\_i, num);

workers.erase(w\_i);

cout << endl;

output\_all(workers, 1);

cout << endl;

break;

}

case 5:

{

list<Worker> out;

int year;

cout << "Введите стаж:" << endl;

cin >> year;

out = search(workers, year);

if (out.size() != 0)

output\_all(out, 1);

else

cout << "Рабочих с указанным стажем не обнаружено" << endl;

cout << endl;

break;

}

case 6:

{

list<Worker> out;

string func;

cout << "Введите должность:" << endl;

cin >> func;

out = search(workers, func);

if(out.size() != 0)

output\_all(out, 1);

else

cout << "Рабочих с данной должностью не обнаружено" << endl;

cout << endl;

break;

}

case 7:

{

string n\_file;

cout << "Введите название файла сохранения" << endl;

cin >> n\_file;

file\_save(workers, n\_file);

cout << endl;

break;

}

case 8:

{

if (workers.size() != 0)

output\_all(workers, 1);

else

{

cout << "Список рабочих пуст" << endl;

break;

}

Worker buf;

int num;

cout << "Выберите номер рабочего, чья запись будет изменена:" << endl;

cin >> num;

cout << endl;

auto w\_i = workers.begin();

advance(w\_i, num);

cout << stand\_out << \*w\_i << endl;

cin >> buf;

mod(workers, buf, num);

cout << endl;

output\_all(workers, 1);

cout << endl;

break;

}

case 9:

{

if (workers.size() != 0)

output\_all(workers, 1);

else

{

cout << "Список рабочих пуст" << endl;

break;

}

int num;

cout << "Выберите тип сортировки:" << endl;

cout << "1. По фамилии." << endl;

cout << "2. По стажу в годах." << endl;

cin >> num;

cout << endl;

sort(workers, num);

output\_all(workers, 1);

cout << endl;

break;

}

case 10:

{

exit = true;

break;

}

default:

break;

}

}

}