

Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное
учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

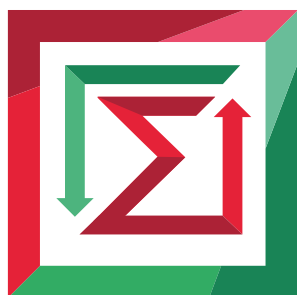


**НГТУ
НЭТИ**

Кафедра теоретической и прикладной информатики

Лабораторная работа № 3
по дисциплине «Объектно-ориентированные технологии разработки
программного обеспечения»

Ассоциативные 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 ассоциативные контейнеры, алгоритмы стандартной библиотеки шаблонов C++ и способы работы с ними.

2. Содержание работы

1. Изучить ассоциативные контейнеры стандартной библиотеки шаблонов C++: множества, мультимножества, отображения, мультиотображения, неупорядоченные множества, неупорядоченные мультимножества, неупорядоченные отображения, неупорядоченные мультиотображения. Изучить алгоритмы стандартной библиотеки шаблонов C++.
2. Переработать программу из работы №2 так, чтобы объекты разработанного в работе №1 класса хранились в неупорядоченном множестве или неупорядоченном мультимножестве. Использовать алгоритмы STL и/или функции-члены контейнера для реализации следующих возможностей:
 - 2.1. Добавление объектов в контейнер.
 - 2.2. Модификация объектов в контейнере (возможно, в соответствии с некоторыми критериями).
 - 2.3. Поиск объектов в контейнере по различным критериям.
 - 2.4. Удаление объектов из контейнера по различным критериям.
 - 2.5. Вывод всех объектов контейнера в файл и/или на экран в отсортированном по какому-либо критерию виде (для сортировки использовать какой-нибудь подходящий вспомогательный контейнер, но не `list` и `forward_list`).
 - 2.6. Распечатка внутренней структуры контейнера с использованием интерфейса сегментов. В программе произвести распечатку как до, так и после повторного хеширования, вызываемого принудительно или автоматически (с использованием установки подходящих значений для минимального количества сегментов и/или максимального коэффициента заполнения и добавления объектов в контейнер), и приводящего к изменению числа сегментов.
3. Для обеспечения возможности создания контейнера разработать хешфункцию и соответствующий ей критерий эквивалентности.
4. При формировании критериев поиска, модификации, удаления использовать функциональные объекты и/или лямбда-выражения.
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.

3. 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

Использованный тип ассоциативного контейнера: unordered_set.

4. Описание алгоритма

Worker(); - конструктор экземпляра класса по умолчанию

```
Worker(const Worker& work) : s_name(work.s_name),  
                             name(work.name),  
                             patronymic(work.patronymic),  
                             function(work.function),  
                             day(work.day),  
                             month(work.month),  
                             year(work.year) {}; - конструктор копирования
```

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_load(unordered_set<Worker, c_hash, c_equal> & workers, string filename)` - функция получения записей из файла

`void file_save(unordered_set<Worker, c_hash, c_equal>& workers, string filename)`) - функция сохранения записей в файл

`void output_all(unordered_set<Worker, c_hash, c_equal>& workers, int format)` - функция вывода всех записей

`unordered_set<Worker, c_hash, c_equal> search(unordered_set<Worker, c_hash, c_equal>& workers, string function)` - функция поиска рабочих по должности

`unordered_set<Worker, c_hash, c_equal> search(unordered_set<Worker, c_hash, c_equal> workers, int year)` - функция поиска рабочих по стажу

`set<Worker, Worker::sort_equal> sort(unordered_set<Worker, c_hash, c_equal> workers, int num)` - функция сортировки списка рабочих

`void mod(unordered_set<Worker, c_hash, c_equal>& workers, Worker buf, int num)` - функция модификации записи рабочего

`void printHashTableState(unordered_set<Worker, c_hash, c_equal>& cont)` - функция вывода внутренней структуры контейнера

5. Примеры работы программы

Рабочее меню программы

```
Выберите функцию:
1. Вывод списка рабочих.
2. Добавление нового рабочего.
3. Добавление рабочих из файла.
4. Удаление рабочих.
5. Поиск рабочих, чей стаж превышает заданное значение.
6. Поиск рабочих по конкретной должности.
7. Сохранение списка рабочих в указанный файл.
8. Модификация указанной записи рабочего.
9. Сортировка списка.
10. Сортировка списка.
11. Выход.
```

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

```
1
Выберите формат вывода:
1. Стандартный в строку.
2. В формате таблицы.
1
Список рабочих пуст
```

Добавление новой записи в список

```
2
Введите данные нового рабочего в формате: Фамилия Имя Отчество Должность (0.0.0000)
Romanov Nicolay Alecsandrovich tester (1.12.1998)
```

Демонстрация, что запись уже в списке

```
1
Выберите формат вывода:
1. Стандартный в строку.
2. В формате таблицы.
1
Romanov Nicolay Alecsandrovich tester (1.12.1998)
```

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

```
1
Выберите формат вывода:
1. Стандартный в строку.
2. В формате таблицы.
2
0. Second name: Romanov
Name: Nicolay
Patronymic: Alecsandrovich
Position: tester
Beginning of work: 1.12.1998

1. Second name: Eliseev
Name: Alexander
Patronymic: Alexandrovich
Position: accountant
Beginning of work: 20.11.1998

2. Second name: Firisov
Name: Boris
Patronymic: Nikolaevich
Position: administrator
Beginning of work: 3.2.2005

3. Second name: Turok
Name: Ivan
Patronymic: Danilovich
Position: administrator
Beginning of work: 15.5.1990

4. Second name: Kirenko
Name: Yulia
Patronymic: Viktorovna
Position: programmer
Beginning of work: 2.12.2003

5. Second name: Yuriy
Name: Tavish
Patronymic: Romanovich
Position: programmer
Beginning of work: 3.11.1992

6. Second name: Arionova
Name: Mariya
Patronymic: Aleksandrovna
Position: tester
Beginning of work: 27.2.1995
```

Демонстрация удаления выбранной записи

```
4
0. Eliseev Alexander Alexandrovich accountant (20.11.1998)
1. Firisov Boris Nikolaevich administrator (3.2.2005)
2. Turok Ivan Danilovich administrator (15.5.1990)
3. Kirenko Yulia Viktorovna programmer (2.12.2003)
4. Yuriy Tavish Romanovich programmer (3.11.1992)
5. Arionova Mariya Aleksandrovna tester (27.2.1995)
Выберите номер рабочего, чья запись будет удалена:
0
0. Firisov Boris Nikolaevich administrator (3.2.2005)
1. Turok Ivan Danilovich administrator (15.5.1990)
2. Kirenko Yulia Viktorovna programmer (2.12.2003)
3. Yuriy Tavish Romanovich programmer (3.11.1992)
4. Arionova Mariya Aleksandrovna tester (27.2.1995)
```

Поиск рабочих, чей стаж превышает заданное число лет

```
5
Введите стаж:
20
0. Turok Ivan Danilovich administrator (15.5.1990)
1. Yuriy Tavish Romanovich programmer (3.11.1992)
2. Arionova Mariya Aleksandrovna tester (27.2.1995)
```

Поиск сотрудников по их должности

```
Введите должность:
programmer
0. Kirenko Yulia Viktorovna programmer (2.12.2003)
1. Yuriy Tavish Romanovich programmer (3.11.1992)
```

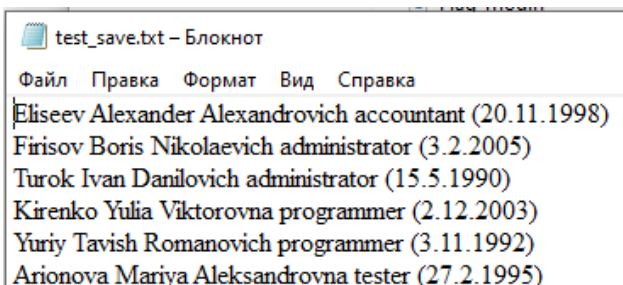
Демонстрация результатов поиска для случаев, когда не найден ни один сотрудник

```
Введите стаж:
40
Рабочих с указанным стажем не обнаружено
```

```
Введите должность:
GOD
Рабочих с данной должностью не обнаружено
```

Сохранение списка рабочих в указанный файл

```
7
Введите название файла сохранения
test_save.txt
```



test_save.txt – Блокнот

Файл Правка Формат Вид Справка

Eliseev Alexander Alexandrovich accountant (20.11.1998)
Firisov Boris Nikolaevich administrator (3.2.2005)
Turok Ivan Danilovich administrator (15.5.1990)
Kirenko Yulia Viktorovna programmer (2.12.2003)
Yuriy Tavish Romanovich programmer (3.11.1992)
Arionova Mariya Aleksandrovna tester (27.2.1995)

Модификация выбранной записи

```
8
0. Eliseev Alexander Alexandrovich accountant (20.11.1998)
1. Firisov Boris Nikolaevich administrator (3.2.2005)
2. Turok Ivan Danilovich administrator (15.5.1990)
3. Kirenko Yulia Viktorovna programmer (2.12.2003)
4. Yuriy Tavish Romanovich programmer (3.11.1992)
5. Arionova Mariya Aleksandrovna tester (27.2.1995)
Выберите номер рабочего, чья запись будет изменена:
0

Eliseev Alexander Alexandrovich accountant (20.11.1998)
TEST TEST TEST TESTER (1.1.2021)

0. TEST TEST TEST TESTER (1.1.2021)
1. Firisov Boris Nikolaevich administrator (3.2.2005)
2. Turok Ivan Danilovich administrator (15.5.1990)
3. Kirenko Yulia Viktorovna programmer (2.12.2003)
4. Yuriy Tavish Romanovich programmer (3.11.1992)
5. Arionova Mariya Aleksandrovna tester (27.2.1995)
```

Сортировка рабочих

По фамилии

```
0. Eliseev Alexander Alexandrovich accountant (20.11.1998)
1. Firisov Boris Nikolaevich administrator (3.2.2005)
2. Turok Ivan Danilovich administrator (15.5.1990)
3. Kirenko Yulia Viktorovna programmer (2.12.2003)
4. Yuriy Tavish Romanovich programmer (3.11.1992)
5. Arionova Mariya Aleksandrovna tester (27.2.1995)
Выберите тип сортировки:
1. По фамилии.
2. По стажу в годах.
1

0. Arionova Mariya Aleksandrovna tester (27.2.1995)
1. Eliseev Alexander Alexandrovich accountant (20.11.1998)
2. Firisov Boris Nikolaevich administrator (3.2.2005)
3. Kirenko Yulia Viktorovna programmer (2.12.2003)
4. Turok Ivan Danilovich administrator (15.5.1990)
5. Yuriy Tavish Romanovich programmer (3.11.1992)
```


По стажу

```
0. Arionova Mariya Aleksandrovna tester (27.2.1995)
1. Eliseev Alexander Alexandrovich accountant (20.11.1998)
2. Firisov Boris Nikolaevich administrator (3.2.2005)
3. Kirenko Yulia Viktorovna programmer (2.12.2003)
4. Turok Ivan Danilovich administrator (15.5.1990)
5. Yuriy Tavish Romanovich programmer (3.11.1992)
Выберите тип сортировки:
1. По фамилии.
2. По стажу в годах.
2

0. Turok Ivan Danilovich administrator (15.5.1990)
1. Yuriy Tavish Romanovich programmer (3.11.1992)
2. Arionova Mariya Aleksandrovna tester (27.2.1995)
3. Eliseev Alexander Alexandrovich accountant (20.11.1998)
4. Kirenko Yulia Viktorovna programmer (2.12.2003)
5. Firisov Boris Nikolaevich administrator (3.2.2005)
```

Вывод внутренней структуры контейнера

До автоматического хеширования

```
size: 6
buckets: 8
load factor: 0.75
max load factor: 1
data:
b[1]: Turok Ivan Danilovich administrator (15.5.1990)
b[3]: Firisov Boris Nikolaevich administrator (3.2.2005)
b[4]: Kirenko Yulia Viktorovna programmer (2.12.2003)
b[5]: Yuriy Tavish Romanovich programmer (3.11.1992)
b[6]: Arionova Mariya Aleksandrovna tester (27.2.1995)
b[7]: Eliseev Alexander Alexandrovich accountant (20.11.1998)
```

После автоматического хеширования (выполняется путём добавления новых записей)

```
size: 12
buckets: 64
load factor: 0.1875
max load factor: 1
data:
b[4]: Chiff Master Ketrinovich restorer (11.11.2004)
b[9]: Yeshua Esus Father God (1.1.2000)
b[17]: Mann Gray Seifenaevich tycoon (30.1.1991)
b[19]: Firisov Boris Nikolaevich administrator (3.2.2005)
b[20]: Lapenko Kirill Ptavovich killer (11.3.2015)
b[30]: Arionova Mariya Aleksandrovna tester (27.2.1995)
b[35]: Diesel Riddick Necromongerovitch infantryman (1.5.1993)
b[38]: Danilenko Sergey Alekseevich creator (1.11.2008)
b[39]: Eliseev Alexander Alexandrovich accountant (20.11.1998)
b[52]: Kirenko Yulia Viktorovna programmer (2.12.2003)
b[57]: Turok Ivan Danilovich administrator (15.5.1990)
b[61]: Yuriy Tavish Romanovich programmer (3.11.1992)
```

6. Текст программы

Файл Worker.h

```
#ifndef WORKER
#define WORKER

#pragma once
#include <iostream>
#include <functional>
#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(const Worker& work) : s_name(work.s_name),
                                     name(work.name),
                                     patronymic(work.patronymic),
                                     function(work.function),
                                     day(work.day),
                                     month(work.month),
                                     year(work.year) {};

        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;
    }

    friend class c_hash;
    friend class c_equal;

    class sort_equal
    {
    public:
        bool mode;
        sort_equal(bool flag = false) : mode(flag) {}

        bool operator() (const Worker& w1, const Worker& w2) const

```

```

        {
            return mode ? w1.s_name < w2.s_name : w1.year <
w2.year;
        }
    };

    };

class c_hash
{
public:
    size_t operator() (const Worker& w) const
    {
        return hash<string>()(w.name) ^
            hash<string>()(w.s_name) ^
            hash<string>()(w.patronymic) ^
            hash<string>()(w.function);
    }

};

class c_equal
{
public:
    bool operator() (const Worker& w1, const Worker& w2) const
    {
        if (w1.name == w2.name && w1.s_name == w2.s_name &&
            w1.patronymic == w2.patronymic)
            return 1;
        return 0;
    }

};

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.setDate(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>
#include <algorithm>
#include <unordered_set>
#include <set>

using namespace std;
using namespace Workers;

void file_load(unordered_set<Worker, c_hash, c_equal> & 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.insert(wr);
    }
    f.close();
};

void file_save(unordered_set<Worker, c_hash, c_equal>& 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(unordered_set<Worker, c_hash, c_equal>& 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++;
}
}

unordered_set<Worker, c_hash, c_equal> search(unordered_set<Worker, c_hash, c_equal>&
workers, string function)
{
    unordered_set<Worker, c_hash, c_equal> out;
    unordered_set<Worker, c_hash, c_equal>::iterator iter;
    iter = find_if(workers.begin(),
workers.end(),
[&](Worker w)
{
    return w.getFunc() == function;
});

    while (iter != workers.end())
    {
        out.insert(*iter);
        iter++;
        iter = find_if(iter, workers.end(), [&](Worker w)
        {
            return w.getFunc() == function;
        });
    }

    return out;
}

unordered_set<Worker, c_hash, c_equal> search(unordered_set<Worker, c_hash, c_equal>
workers, int year)
{
    struct tm* date;
    time_t t = time(NULL);
    date = gmtime(&t);
    int tt = date->tm_year + 1900;
    unordered_set<Worker, c_hash, c_equal> out;
    unordered_set<Worker, c_hash, c_equal>::iterator iter;
    iter = find_if(workers.begin(),
workers.end(),
[&](Worker w)
{
    return (tt - w.getYear()) > year;
});

    while (iter != workers.end())
    {
        out.insert(*iter);
        iter++;
        iter = find_if(iter,
workers.end(),
[&](Worker w)
{
    return (tt - w.getYear()) > year;
        });
    }

    return out;
};

```



```

set<Worker, Worker::sort_equal> sort(unordered_set<Worker, c_hash, c_equal> workers, int
num)
{
    if (num == 1)
    {
        set<Worker, Worker::sort_equal> out(Worker::sort_equal{ true });

        for (auto iter = workers.begin(); iter != workers.end(); iter++)
            out.insert(*iter);
        return out;
    }
    if (num == 2)
    {
        set<Worker, Worker::sort_equal> out(Worker::sort_equal{ false });

        for (auto iter = workers.begin(); iter != workers.end(); iter++)
            out.insert(*iter);
        return out;
    }
}

void mod(unordered_set<Worker, c_hash, c_equal>& workers, Worker buf, int num)
{
    auto w_i = workers.begin();
    advance(w_i, num);
    workers.erase(w_i);
    buf.setS_name(buf.getS_name());
    buf.setName(buf.getName());
    buf.setPatr(buf.getPatr());
    buf.setFunc(buf.getFunc());
    buf.setDate(buf.getDay(), buf.getMonth(), buf.getYear());
    workers.insert(buf);
};

void printHashTableState(unordered_set<Worker, c_hash, c_equal>& cont)
{
    // основные данные о структуре
    cout << "size: " << cont.size() << endl;
    cout << "buckets: " << cont.bucket_count() << endl;
    cout << "load factor: " << cont.load_factor() << endl;
    cout << "max load factor: " << cont.max_load_factor() << endl;
    // элементы в сегментах
    cout << "data: " << endl;
    for (int idx = 0; idx != cont.bucket_count(); ++idx) if (cont.bucket_size(idx) > 0) {
        cout << " b[" << idx << "]: ";
        for (auto pos = cont.begin(idx); pos != cont.end(idx); ++pos)
            cout << *pos << " ";
        cout << endl;
    }
    cout << endl;
}

int main()
{
    setlocale(LC_ALL, "Russian");

    bool exit = false;
    int option;

    /*
    unordered_set<Worker, c_hash, c_equal> 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. Сортировка списка." << endl;
cout << "11. 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;

    workers.insert(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:
{
    unordered_set<Worker, c_hash, c_equal> 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:
{
    unordered_set<Worker, c_hash, c_equal> 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;
        }
        set<Worker, Worker::sort_equal> sort_w;
        int num;
        cout << "Выберите тип сортировки:" << endl;
        cout << "1. По фамилии." << endl;
        cout << "2. По стажу в годах." << endl;
        cin >> num;
        cout << endl;
        sort_w = sort(workers, num);

        int i = 0;
        for (auto iter = sort_w.begin(); iter != sort_w.end(); iter++)
        {
            cout << stand_out << i << ". " << *iter << endl;
            i++;
        }
        cout << endl;
        break;
    }
    case 10:
    {
        if (workers.size() != 0)
        {
            printHashTableState(workers);
            cout << "Добавим новые данные в список рабочих" << endl;
            file_load(workers, "new_data.txt");
            cout << endl;
            printHashTableState(workers);
        }
        else
        {
            cout << "Список рабочих пуст" << endl;
            break;
        }

        break;
    }
    case 11:
    {
        exit = true;
        break;
    }
    default:
        break;
    }
}
}

```