

Звіт

Автор: Богданов І.Ю. КІТ-119а Дата: 17 квітня 2020

Лабораторна робота №14. СОРТУВАННЯ

Тема. STL. Алгоритми зміни послідовності. Сортування. Функтори.

Мета: на практиці порівняти STL-алгоритми, що модифікують послідовність; отримати навички роботи з STL-функторами.

1. Завдання до роботи **Індивідуальне завдання:**

Поширити попередню лабораторну роботу додавши функції об'єднання двох контейнерів типу `map` та, вектор. І додавши функцію сортування до тих контейнерів в яких це можливо.

2. Опис класів, змінних, методів та функцій

2.1 Опис класів

Базовий клас: `file`

Клас нащадок базового класу: `executable_file` і `non_executable_file`

Клас, що має в собі масив нащадків базового класу та методи для роботи з ними: `dir`

Клас, що повинен демонструвати композицію: `options`

2.2 Опис змінних

`std::vector<file> vect` – контейнер типу вектор з елементами базового класу.

`std::list<file> lst` – контейнер типу список з елементами базового класу.

`std::map<int, file> mp` – контейнер типу `map` з елементами базового класу.

`std::set<file> st` – контейнер типу `set` з елементами базового класу.

`int n, w` – службові змінні необхідні для реалізації вибору пунктів меню.

`size_t_t size` – поле класу `file` та його нащадків(розмір файлу у бітах).

`size_t_t` `index` – поле класу `file` та його нащадків(унікальний індекс).

`bool` `is_system` – поле класу `options` та його нащадків(чи є файл системним).

`bool` `is_hidden` – поле класу `options` та його нащадків(чи є файл прихованим).

`std::string` `name` – поле класу `file` та його нащадків(назва файлу).

`size_t_t` `x` – поле класу `file` та його нащадків(64 або 32 розрядна програма).

`options` `opt` – поле класу `file` та його нащадків(чи є файл прихованим і системним).

`int` `next_ind` – поле класу `dir`(номер наступного файлу у директорії).

`int` `new_ind` – поле класу `dir`(індекс наступного файлу у директорії).

`file**` `files` – поле класу `dir`(масив елементів класу `file`).

`file**` `copy` – поле класу `dir`(показчик на клас `file`, використовується для правильної роботи деяких методів).

`bool` `is_text` – поле класу `non_executable_file`(чи є даний файл текстовим).

`size_t_t` `runningtime` – поле класу `executable_file`(час виконання програми).

2.3 Опис методів

Зауваження: класи нащадки мають усі методи класу `file`.

`void` `change_ver(const int&)` – зміна значення поля `ver` змінної класу `x_version`(метод класу `x_version`).

`virtual size_t_t` `get_x()` `const` – отримання значення поля `x` змінної класу `file`(метод класу `file`).

`virtual size_t_t` `get_size()` `const` – отримання значення поля `size` змінної класу `file`(метод класу `file`).

`virtual size_t_t` `get_index()` `const` – отримання значення поля `index` змінної класу `file`(метод класу `file`).

`virtual bool` `get_sys()` `const` – отримання значення поля `is_system` змінної класу `file`(метод класу `file`).

`virtual bool` `get_hid()` `const` – отримання значення поля `is_hidden` змінної класу `file`(метод класу `file`).

`virtual std::string get_name() const` – отримання значення поля `name` змінної класу `file` (метод класу `file`).

`virtual void change_x(const size_t_t &x)` – зміна значення поля `x` змінної класу `file` (метод класу `file`).

`virtual void change_size(const size_t_t &sz)` – зміна значення поля `size` змінної класу `file` (метод класу `file`).

`virtual void change_index(const size_t_t &in)` – зміна значення поля `index` змінної класу `file` (метод класу `file`).

`virtual void change_sys(const bool&)` – зміна значення поля `is_system` змінної класу `file` (метод класу `file`).

`virtual void change_hid(const bool&)` – зміна значення поля `is_hidden` змінної класу `file` (метод класу `file`).

`virtual void change_name(const std::string&)` – зміна значення поля `name` змінної класу `file` (метод класу `file`).

`file()` – конструктор класу `file`.

`file(const file&)` – конструктор копіювання класу `file`.

`file(const int&, const int&, const int&, const bool&, const bool&, const std::string&)` – конструктор з параметрами класу `file`.

`~file()` – деструктор класу `file`.

`void add_file(const file &f)` – додавання об'єкту класу `file` до масиву в класі `dir` (метод класу `dir`).

`void del_file(const int &index)` – видалення об'єкту класу `file` з масиву в класі `dir` (метод класу `dir`).

`void del_all()` – видалення усіх об'єктів класу `file` з масиву в класі `dir` (метод класу `dir`).

`void add_file_by_str(const std::string& str, x_version*, x_version*)` – додавання об'єкту класу `file` до масиву в класі `dir` за допомогою строки з інформацією про об'єкт (метод класу `dir`).

`void read_from_file(const std::string& name, x_version*, x_version*)` – заповнення масиву об'єктів класу `file` інформація про які буде зчитана з файлу (метод класу `dir`).

`file get_file_by_index(const int& index) const` – отримання об'єкту класу `file` з масиву в класі `dir` (метод класу `dir`).

`void get_file_to_screen(const int &index) const` – виведення об'єкту класу `file` з масиву в класі `dir` на екран (метод класу `dir`).

`void print_all() const` – виведення усіх об'єктів класу `file` з масиву в класі `dir` на екран (метод класу `dir`).

`int count_system() const` – розрахування кількості скритих і системних файлів в об'єкті класу `dir` (метод класу `dir`).

`void print_to_file(const std::string& name) const` – запис у файл інформації про об'єкти класу `file` що є в масиві (метод класу `dir`).

`void get_file_to_screen(const int &index) const` – запис у рядок інформації про об'єкт класу `file` (метод класу `dir`).

`bool check_str(const std::string& str) const` – перевірка рядка на відповідність формату зберігання даних про об'єкт класу `file` (метод класу `dir`).

`void print_all_with_2_or_more_words() const` – виведення усіх об'єктів класу `file` в назві яких є 2 або більше слів з масиву в класі `dir` на екран(метод класу `dir`).

`void sort_files(bool (*f)(file&,file&))` – сортування усіх об'єктів класу `file` в об'єкті класу `dir` на екран(метод класу `dir`).

`bool get_sys() const` – отримання значення поля `is_system` змінної класу `options`(метод класу `options`).

`bool get_hid() const` – отримання значення поля `is_hidden` змінної класу `options`(метод класу `options`).

`void change_sys(const bool&)` – зміна значення поля `is_system` змінної класу `options`(метод класу `options`).

`void change_hid(const bool&)` – зміна значення поля `is_hidden` змінної класу `options`(метод класу `options`).

`executable_file()` – конструктор класу `executable_file`.

`executable_file(const executable_file&)` – конструктор копіювання класу `executable_file`.

`executable_file(x_version*, const size_t_t&, const size_t_t&, const bool&, const bool&, const std::string&, const size_t_t&)` – конструктор з параметрами класу `executable_file`.

`~executable_file()` – деструктор класу `executable_file`.

`non_executable_file()` – конструктор класу `non_executable_file`.

`non_executable_file(const executable_file&)` – конструктор копіювання класу `non_executable_file`.

`non_executable_file(x_version*, const size_t_t&, const size_t_t&, const bool&, const bool&, const std::string&, const bool&)` – конструктор з параметрами класу `non_executable_file`.

`~non_executable_file()` – деструктор класу `non_executable_file`.

`virtual std::string get_info() const = 0` – віртуальний метод базового класу. В класах нащадках перевантажений на виведення інформації, про об'єкт класу нащадку, яка є специфічною саме для цього класу-нащадку.

`virtual size_t_t get_time() const final` – метод класу `executable_file`, повертає значення поля `runningtime`.

`virtual void change_time(const size_t_t&) final` – метод класу `executable_file`, змінює значення поля `runningtime`.

`virtual bool get_text() const final` – метод класу `non_executable_file`, повертає значення поля `is_text`.

`virtual void change_text(const bool&) final` – метод класу `non_executable_file`, змінює значення поля `is_text`.

2.4 Опис функцій

`void menu()` – функція меню.

`void old_menu()` – функція меню.

`bool sort_version(file&, file&)` – функція порівняння двох файлів по х.

`bool sort_size(file&, file&)` – функція порівняння двох файлів по розміру.

`bool sort_ind(file&, file&)` – функція порівняння двох файлів по індексу.

`bool sort_opt(file&, file&)` – функція порівняння двох файлів по тому чи ж вони системними або прихованими.

`bool sort_name(file&, file&)` – функція порівняння двох файлів по назві.

`bool check_str(const std::string& str)` – функція перевірки строки на відповідність формату назви файлу.

`bool operator==(const file& f1, const file& f2)` – перевантаження оператора порівняння.

`bool operator!=(const file& f1, const file& f2)` – перевантаження ще одного оператора порівняння.

`bool operator==(const executable_file& f1, const executable_file& f2)` – аналогічне перевантаження для класу нащадку.

`bool operator!=(const executable_file& f1, const executable_file& f2)` – аналогічне перевантаження для класу нащадку.

`bool operator==(const non_executable_file& f1, const non_executable_file& f2)` – аналогічне перевантаження для класу нащадку.

`bool operator!=(const non_executable_file& f1, const non_executable_file& f2)` – аналогічне перевантаження для класу нащадку.

`std::ostream& operator<<(std::ostream& os, const executable_file& f)` – перевантаження оператора виведення.

`std::ostream& operator<<(std::ostream& os, const non_executable_file& f)` – аналогічне перевантаження оператора виведення.

`std::istream& operator>>(std::istream& is, executable_file& f)` – перевантаження оператора введення.

`std::istream& operator>>(std::istream& is, non_executable_file& f)` – перевантаження оператора введення.

`std::ostream& operator<<(std::ostream& os, const file& f)` – аналогічне перевантаження оператора виведення.

3 Текст програми

Лабораторная работа 14.cpp

```
#include "menu.h"
```

```
int main(){  
    menu();  
}
```

file.h

```
#pragma once  
#define _CRT_SECURE_NO_WARNINGS  
#include <iostream>  
#include <sstream>  
#include <fstream>  
#include <string>  
#include <regex>  
#include <iomanip>  
#include <vector>
```

```

#include <list>
#include <set>
#include <map>
#include <algorithm>

typedef size_t size_t_t;

class options {
private:
    bool issystem;
    bool ishidden;
public:
    bool get_sys() const;
    bool get_hid() const;
    void change_sys(const bool&);
    void change_hid(const bool&);
};

class file {
protected:
    size_t_t x;
    size_t_t size;
    size_t_t index;
    options opt;
    std::string name;
public:
    int type_of_file = 0;
    virtual size_t_t get_x() const;
    virtual size_t_t get_size() const;
    virtual size_t_t get_index() const;
    virtual bool get_sys() const;
    virtual bool get_hid() const;
    virtual std::string get_name() const;
    virtual void change_x(const size_t_t&);
    virtual void change_size(const size_t_t&);
    virtual void change_index(const size_t_t&);
    virtual void change_sys(const bool&);
    virtual void change_hid(const bool&);
    virtual void change_name(const std::string&);
    virtual std::string get_info() const;
    file();
    file(const file&);
    file(const size_t_t&, const size_t_t&, const size_t_t&, const bool&, const bool&, const
std::string&);
    ~file();
    file& operator=(const file& f);
    friend bool operator==(const file& f1, const file& f2);
    friend bool operator!=(const file& f1, const file& f2);
};

class executable_file final : public file {
private:
    size_t_t runningtime;
public:
    virtual size_t_t get_time() const final;
    virtual void change_time(const size_t_t&) final;
    executable_file();
    executable_file(const executable_file&);
    executable_file(const size_t_t&, const size_t_t&, const size_t_t&, const bool&, const bool&,
const std::string&, const size_t_t&);
    ~executable_file();
    virtual std::string get_info() const final;
    executable_file& operator=(const executable_file& f);
    friend bool operator==(const executable_file& f1, const executable_file& f2);
    friend bool operator!=(const executable_file& f1, const executable_file& f2);
};

class non_executable_file final : public file {
private:
    bool is_text;
public:
    virtual bool get_text() const final;

```

```

        virtual void change_text(const bool&) final;
        non_executable_file();
        non_executable_file(const non_executable_file&);
        non_executable_file(const size_t_t&, const size_t_t&, const size_t_t&, const bool&, const
bool&, const std::string&, const bool&);
        ~non_executable_file();
        virtual std::string get_info() const final;
        non_executable_file& operator=(const non_executable_file& f);
        friend bool operator==(const non_executable_file& f1, const non_executable_file& f2);
        friend bool operator!=(const non_executable_file& f1, const non_executable_file& f2);
};

bool check_str(const std::string& str);
bool operator==(const file& f1, const file& f2);
bool operator!=(const file& f1, const file& f2);
bool operator==(const executable_file& f1, const executable_file& f2);
bool operator!=(const executable_file& f1, const executable_file& f2);
bool operator==(const non_executable_file& f1, const non_executable_file& f2);
bool operator!=(const non_executable_file& f1, const non_executable_file& f2);
bool operator>(const file& f1, const file& f2);
bool operator<(const file& f1, const file& f2);
bool operator>(const executable_file& f1, const executable_file& f2);
bool operator<(const executable_file& f1, const executable_file& f2);
bool operator>(const non_executable_file& f1, const non_executable_file& f2);
bool operator<(const non_executable_file& f1, const non_executable_file& f2);
std::ostream& operator<<(std::ostream& os, const executable_file& f);
std::ostream& operator<<(std::ostream& os, const non_executable_file& f);
std::istream& operator>>(std::istream& is, executable_file& f);
std::istream& operator>>(std::istream& is, non_executable_file& f);
std::ostream& operator<<(std::ostream& os, const file& f);

```

file.cpp

```

#include "file.h"

size_t_t file::get_x() const {
    return x;
}

size_t_t file::get_size() const {
    return size;
}

size_t_t file::get_index() const {
    return index;
}

bool file::get_sys() const {
    return opt.get_sys();
}

bool file::get_hid() const {
    return opt.get_hid();
}

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

void file::change_x(const size_t_t& new_x) {
    x = new_x;
}

void file::change_size(const size_t_t& sz) {
    size = sz;
}

void file::change_index(const size_t_t& in) {
    index = in;
}

void file::change_sys(const bool& sys) {
    opt.change_sys(sys);
}

void file::change_hid(const bool& hid) {
    opt.change_hid(hid);
}

void file::change_name(const std::string& nm) {
    name = nm;
}

```



```

file::file() {
    x = 0;
    size = 100;
    index = 0;
    opt.change_hid(false);
    opt.change_sys(false);
    name = "File";
    std::cout << "Файл создан при помощи конструктора по умолчанию." << "\n";
}
file::file(const file& f) {
    x = f.x;
    size = f.size;
    index = f.index;
    opt = f.opt;
    name = f.name;
}
file::file(const size_t& ver, const size_t& sz, const size_t& ind, const bool& sys, const
bool& hid, const std::string& nm) {
    x = ver;
    size = sz;
    index = ind;
    opt.change_sys(sys);
    opt.change_hid(hid);
    name = nm;
    std::cout << "Файл создан при помощи конструктора с аргументами." << "\n";
}
file::~file() {
    std::cout << "Файл уничтожен при помощи деструктора по умолчанию." << "\n";
}
bool options::get_sys() const {
    return issystem;
}
bool options::get_hid() const {
    return ishidden;
}
void options::change_sys(const bool& sys) {
    issystem = sys;
}
void options::change_hid(const bool& hid) {
    ishidden = hid;
}
size_t executable_file::get_time() const {
    return runningtime;
}
executable_file::executable_file() : file(), runningtime(0) {
    type_of_file = 1;
}
executable_file::executable_file(const executable_file& sf) : file(sf), runningtime(sf.runningtime)
{
    type_of_file = 1;
}
executable_file::executable_file(const size_t& x, const size_t& sz, const size_t& ind, const
bool& sys, const bool& hid, const std::string& nm, const size_t& time) : file(x, sz, ind, sys,
hid, nm), runningtime(time) {
    type_of_file = 1;
}
executable_file::~executable_file() {}
void executable_file::change_time(const size_t& time) {
    runningtime = time;
}
bool non_executable_file::get_text() const {
    return is_text;
}
non_executable_file::non_executable_file() : file(), is_text(true) {
    type_of_file = 2;
}
non_executable_file::non_executable_file(const non_executable_file& nf) : file(nf),
is_text(nf.is_text) {
    type_of_file = 2;
}

```



```

}
non_executable_file::non_executable_file(const size_t& x, const size_t& sz, const size_t& ind,
const bool& sys, const bool& hid, const std::string& nm, const bool& text) : file(x, sz, ind, sys,
hid, nm), is_text(text) {
    type_of_file = 2;
}
non_executable_file::~non_executable_file() {}
void non_executable_file::change_text(const bool& text) {
    is_text = text;
}
std::string executable_file::get_info() const {
    std::stringstream s;
    s << runningtime;
    return s.str();
}
std::string non_executable_file::get_info() const {
    std::stringstream s;
    s << is_text;
    return s.str();
}
bool operator==(const file& f1, const file& f2) {
    if (f1.get_name() != f2.get_name()) {
        return false;
    }
    else if (f1.get_sys() != f2.get_sys()) {
        return false;
    }
    else if (f1.get_size() != f2.get_size()) {
        return false;
    }
    else if (f1.get_hid() != f2.get_hid()) {
        return false;
    }
    else if (f1.get_x() != f2.get_x()) {
        return false;
    }
    else if (f1.get_index() != f2.get_index()) {
        return false;
    }
    else {
        return true;
    }
}
bool operator!=(const file& f1, const file& f2) {
    return !(f1 == f2);
}
bool operator==(const executable_file& f1, const executable_file& f2) {
    if (f1.get_name() != f2.get_name()) {
        return false;
    }
    else if (f1.get_sys() != f2.get_sys()) {
        return false;
    }
    else if (f1.get_size() != f2.get_size()) {
        return false;
    }
    else if (f1.get_hid() != f2.get_hid()) {
        return false;
    }
    else if (f1.get_x() != f2.get_x()) {
        return false;
    }
    else if (f1.get_index() != f2.get_index()) {
        return false;
    }
    else if (f1.get_time() != f2.get_time()) {
        return false;
    }
    else {

```

```

        return true;
    }
}
bool operator!=(const executable_file& f1, const executable_file& f2) {
    return !(f1 == f2);
}
bool operator==(const non_executable_file& f1, const non_executable_file& f2) {
    if (f1.get_name() != f2.get_name()) {
        return false;
    }
    else if (f1.get_sys() != f2.get_sys()) {
        return false;
    }
    else if (f1.get_size() != f2.get_size()) {
        return false;
    }
    else if (f1.get_hid() != f2.get_hid()) {
        return false;
    }
    else if (f1.get_x() != f2.get_x()) {
        return false;
    }
    else if (f1.get_index() != f2.get_index()) {
        return false;
    }
    else if (f1.get_text() != f2.get_text()) {
        return false;
    }
    else {
        return true;
    }
}
bool operator!=(const non_executable_file& f1, const non_executable_file& f2) {
    return !(f1 == f2);
}
bool check_str(const std::string& str){
    std::regex re("[A-Za-zA-Яa-я0-9\\s\\!,\\?\\\"\\.\\.;\\']*");
    if (!(std::regex_search(str, re))) {
        return false;
    }
    std::regex re_2("\\s{2,}");
    if (std::regex_search(str, re_2)) {
        return false;
    }
    std::regex re_3("[\\!\\?:\\.\\,\\;]{2,}");
    if (std::regex_search(str, re_3)) {
        return false;
    }
    std::regex re_4("[\\'\\\"]{2,}");
    if (std::regex_search(str, re_4)) {
        return false;
    }
    return true;
}
std::ostream& operator<<(std::ostream& os, const file& f) {
    return os << f.type_of_file << " " << "\"" << f.get_name() << "\" " << f.get_index() << " "
<< f.get_size() << " " << f.get_x() << " " << f.get_hid() << " " << f.get_sys() << " " <<
f.get_info();
}
std::ostream& operator<<(std::ostream& os, const executable_file& f) {
    return os << f.type_of_file << " " << "\"" << f.get_name() << "\" " << f.get_index() << " "
<< f.get_size() << " " << f.get_x() << " " << f.get_hid() << " " << f.get_sys() << " " <<
f.get_time();
}
std::ostream& operator<<(std::ostream& os, const non_executable_file& f) {
    return os << f.type_of_file << " " << "\"" << f.get_name() << "\" " << f.get_index() << " "
<< f.get_size() << " " << f.get_x() << " " << f.get_hid() << " " << f.get_sys() << " " <<
f.get_text();
}

```

```

std::istream& operator>>(std::istream& is, executable_file& f) {
    std::string name;
    std::string temp;
    std::regex re("\\$");
    std::stringstream temps;
    executable_file tempf;
    bool check = true;
    bool global_check = true;
    do {
        is >> temp;
        if (check_str(temp)) {
            name += temp;
        }
        else {
            global_check = false;
        }
        if (std::regex_search(name, re)) {
            check = false;
        }
        else {
            name += " ";
        }
    } while (check);
    std::regex r("\\");
    name = std::regex_replace(name, r, "");
    tempf.change_name(name);
    int temp_i = 0;
    is >> temp;
    if (!check_str(temp)) {
        global_check = false;
    }
    temps << temp;
    temps >> temp_i;
    temps.clear();
    tempf.change_index(temp_i);
    is >> temp;
    if (!check_str(temp)) {
        global_check = false;
    }
    temps << temp;
    temps >> temp_i;
    temps.clear();
    tempf.change_size(temp_i);
    is >> temp;
    if (!check_str(temp)) {
        global_check = false;
    }
    temps << temp;
    temps >> temp_i;
    temps.clear();
    tempf.change_x(temp_i);
    is >> temp;
    if (!check_str(temp)) {
        global_check = false;
    }
    temps << temp;
    temps >> temp_i;
    temps.clear();
    tempf.change_hid(temp_i);
    is >> temp;
    if (!check_str(temp)) {
        global_check = false;
    }
    temps << temp;
    temps >> temp_i;
    temps.clear();
    tempf.change_sys(temp_i);
    is >> temp;
    if (!check_str(temp)) {

```

```

        global_check = false;
    }
    temps << temp;
    temps >> temp_i;
    temps.clear();
    tempf.change_time(temp_i);
    if (global_check == true) {
        f = tempf;
    }
    else {
        f.type_of_file = -1;
    }
    return is;
}
std::istream& operator>>(std::istream& is, non_executable_file& f) {
    std::string name;
    std::string temp;
    std::regex re("\\$");
    std::stringstream temps;
    non_executable_file tempf;
    bool check = true;
    bool global_check = true;
    do {
        is >> temp;
        if (check_str(temp)) {
            name += temp;
        }
        else {
            global_check = false;
        }
        if (std::regex_search(name, re)) {
            check = false;
        }
        else {
            name += " ";
        }
    } while (check);
    std::regex r("\\");
    name = std::regex_replace(name, r, "");
    tempf.change_name(name);
    int temp_i = 0;
    is >> temp;
    if (!check_str(temp)) {
        global_check = false;
    }
    temps << temp;
    temps >> temp_i;
    temps.clear();
    tempf.change_index(temp_i);
    is >> temp;
    if (!check_str(temp)) {
        global_check = false;
    }
    temps << temp;
    temps >> temp_i;
    temps.clear();
    tempf.change_size(temp_i);
    is >> temp;
    if (!check_str(temp)) {
        global_check = false;
    }
    temps << temp;
    temps >> temp_i;
    temps.clear();
    tempf.change_x(temp_i);
    is >> temp;
    if (!check_str(temp)) {
        global_check = false;
    }
}

```

```

        temps << temp;
        temps >> temp_i;
        temps.clear();
        tempf.change_hid(temp_i);
        is >> temp;
        if (!check_str(temp)) {
            global_check = false;
        }
        temps << temp;
        temps >> temp_i;
        temps.clear();
        tempf.change_sys(temp_i);
        is >> temp;
        if (!check_str(temp)) {
            global_check = false;
        }
        temps << temp;
        temps >> temp_i;
        temps.clear();
        tempf.change_text(temp_i);
        if (global_check == true) {
            f = tempf;
        }
        else {
            f.type_of_file = -1;
        }
        return is;
    }
}

file& file::operator=(const file& f) {
    x = f.x;
    size = f.size;
    index = f.index;
    name = f.name;
    opt.change_sys(f.get_sys());
    opt.change_hid(f.get_hid());
    return *this;
}

executable_file& executable_file::operator=(const executable_file& f) {
    x = f.x;
    size = f.size;
    index = f.index;
    name = f.name;
    opt.change_sys(f.get_sys());
    opt.change_hid(f.get_hid());
    runningtime = f.runningtime;
    return *this;
}

non_executable_file& non_executable_file::operator=(const non_executable_file& f) {
    x = f.x;
    size = f.size;
    index = f.index;
    name = f.name;
    opt.change_sys(f.get_sys());
    opt.change_hid(f.get_hid());
    is_text = f.is_text;
    return *this;
}

std::string file::get_info() const {
    return "";
}

bool operator>(const file& f1, const file& f2) {
    return f1.get_name() < f2.get_name();
}

bool operator<(const file& f1, const file& f2) {
    return f1.get_name() > f2.get_name();
}

bool operator>(const executable_file& f1, const executable_file& f2) {
    return f1.get_name() < f2.get_name();
}

```

```

bool operator<(const executable_file& f1, const executable_file& f2) {
    return f1.get_name() > f2.get_name();
}
bool operator>(const non_executable_file& f1, const non_executable_file& f2) {
    return f1.get_name() < f2.get_name();
}
bool operator<(const non_executable_file& f1, const non_executable_file& f2) {
    return f1.get_name() > f2.get_name();
}

```

dir.h

```

#pragma once
#include "file.h"

class dir { /** Клас - массив. */
private:
    file** files;
    file** copy;
    int next_ind = 0;
    int new_ind = 1;
public:
    void add_file(const executable_file& f);
    void add_file(const non_executable_file& f);
    void del_file(const int& index);
    void del_all();
    void add_file_by_str(const std::string& str);
    void read_from_file(const std::string& name);
    std::string get_file_to_str(const int& index) const;
    file* get_file_by_index(const int& index) const;
    void print_all() const;
    int count_system() const;
    void print_to_file(const std::string& name) const;
    bool check_str(const std::string& str) const;
    void print_all_with_2_or_more_words() const;
    void sort_files(bool (*f)(file&, file&));
    file* operator[](const int& index) {
        return files[index];
    }
    friend std::ostream& operator<<(std::ostream& on, dir& d);
};

struct count_64 {
    count_64() { count = 0; }
    void operator()(file f);
    int count;
};

struct count_32 {
    count_32() { count = 0; }
    void operator()(file f);
    int count;
};

struct count_sys {
    count_sys() { count = 0; }
    void operator()(file f);
    int count;
};

struct count_hid {
    count_hid() { count = 0; }
    void operator()(file f);
    int count;
};

struct ccount_64 {
    ccount_64() { count = 0; }
    void operator()(std::pair<int, file> p);
    int count;
};

struct ccount_32 {
    ccount_32() { count = 0; }
    void operator()(std::pair<int, file> p);
    int count;
};

```

```

};
struct ccount_sys {
    ccount_sys() { count = 0; }
    void operator()(std::pair<int, file> p);
    int count;
};
struct ccount_hid {
    ccount_hid() { count = 0; }
    void operator()(std::pair<int, file> p);
    int count;
};
struct sort_by {
    sort_by(int i) { type = i; }
    bool operator()(file& f1, file& f2);
    int type;
};

bool sort_version(file&, file&);
bool sort_size(file&, file&);
bool sort_ind(file&, file&);
bool sort_opt(file&, file&);
bool sort_name(file&, file&);
std::istream& operator>>(std::istream& is, dir& d);
std::ostream& operator<<(std::ostream& os, dir& d);
void print(const file& f);
void pprint(const std::pair<int, file>& p);
void print_64(const file& f);
void print_32(const file& f);
void print_sys(const file& f);
void print_hid(const file& f);
void pprint_64(const std::pair<int, file>& p);
void pprint_32(const std::pair<int, file>& p);
void pprint_sys(const std::pair<int, file>& p);
void pprint_hid(const std::pair<int, file>& p);

```

dir.cpp

```
#include "dir.h"
```

```

void dir::add_file(const executable_file& f) {
    if (next_ind == 0) {
        files = new file * [next_ind + 1];
        file* ptr = new auto(f);
        files[next_ind] = ptr;
        new_ind++;
        next_ind++;
    }
    else {
        copy = new file * [next_ind + 1];
        for (int i = 0; i < next_ind; i++) {
            copy[i] = files[i];
        }
        delete[] files;
        files = new file * [next_ind + 1];
        for (int i = 0; i < next_ind; i++) {
            files[i] = copy[i];
        }
        file* ptr = new auto(f);
        files[next_ind] = ptr;
        delete[] copy;
        next_ind++;
        new_ind++;
    }
}

void dir::add_file(const non_executable_file& f) {
    if (next_ind == 0) {
        files = new file * [next_ind + 1];
        file* ptr = new auto(f);
        files[next_ind] = ptr;
        new_ind++;
    }
}

```



```

        next_ind++;
    }
    else {
        copy = new file * [next_ind + 1];
        for (int i = 0; i < next_ind; i++) {
            copy[i] = files[i];
        }
        delete[] files;
        files = new file * [next_ind + 1];
        for (int i = 0; i < next_ind; i++) {
            files[i] = copy[i];
        }
        file* ptr = new auto(f);
        files[next_ind] = ptr;
        delete[] copy;
        next_ind++;
        new_ind++;
    }
}

void dir::del_file(const int& index) {
    if (next_ind == 1) {
        delete[] files;
        next_ind--;
    }
    else {
        copy = new file * [next_ind];
        for (int i = 0; i < index; i++) {
            copy[i] = files[i];
        }
        for (int i = index; i < next_ind; i++) {
            copy[i] = files[i + 1];
        }
        delete files[index];
        delete[] files;
        files = copy;
        copy = 0;
    }
}

void dir::del_all() {
    if (next_ind != 0) {
        for (int i = 0; i < next_ind; i++) {
            delete files[i];
        }
        delete[] files;
        next_ind = 0;
    }
}

void dir::read_from_file(const std::string& name) {
    del_all();
    std::ifstream f;
    char* temp;
    f.open(name);
    while (!f.eof()) {
        temp = new char[100];
        f.getline(temp, 100);
        add_file_by_str(temp);
        delete[] temp;
    }
    f.close();
}

std::string dir::get_file_to_str(const int& index) const {
    std::stringstream s;
    s << files[index];
    return s.str();
}

void dir::print_all() const {
    for (int i = 0; i < next_ind; i++) {
        std::cout << i + 1 << " " << (*(files+i)) << "\n";
    }
}

```

```

}
int dir::count_system() const {
    auto count = 0;
    for (int i = 0; i < next_ind; i++) {
        if (files[i]->get_hid() && files[i]->get_sys()) {
            count++;
        }
    }
    return count;
}
file* dir::get_file_by_index(const int& index) const {
    for (int i = 0; i < next_ind; i++) {
        if (files[i]->get_index() == index) {
            return files[i];
        }
    }
}
void dir::add_file_by_str(const std::string& str) {
    if (check_str(str)) {
        std::stringstream s;
        std::regex re("\\.+"");
        std::smatch sm;
        std::string temp;
        std::regex_search(str, sm, re);
        auto i = str.find("\\");
        i = str.find("\\", i + 1);
        std::regex re_temp("\\");
        temp = sm[0];
        std::string name = std::regex_replace(temp, re_temp, "");
        auto i2 = str.find(" ", i + 2);
        temp = str.substr(i + 1, i2 - i);
        s << temp;
        auto index = 0;
        int check = 0;
        s >> check;
        s.clear();
        i = i2;
        i2 = str.find(" ", i + 1);
        temp = str.substr(i + 1, i2 - i);
        s << temp;
        s >> index;
        s.clear();
        int i3 = str.find(" ", i2 + 1);
        s.clear();
        temp = str.substr(i2 + 1, i3 - i2);
        s << temp;
        auto size = 0;
        s >> size;
        auto i4 = str.find(" ", i3 + 1);
        s.clear();
        temp = str.substr(i3 + 1, i4 - i3);
        s << temp;
        auto x = 0;
        s >> x;
        auto i5 = str.find(" ", i4 + 1);
        s.clear();
        temp = str.substr(i4 + 1, i5 - i4);
        s << temp;
        auto hid = false;
        s >> hid;
        auto i6 = str.find(" ", i5 + 1);
        s.clear();
        temp = str.substr(i5 + 1, i6 - i5);
        s << temp;
        auto sys = false;
        s >> sys;
        auto i7 = str.find(" ", i6 + 1);
        temp = str.substr(i6 + 1, i7 - i6);
        s.clear();
    }
}

```

```

        s << temp;
        if (check == 1) {
            size_t_t time;
            s >> time;
            executable_file new_file(x, size, index, sys, hid, name, time);
            add_file(new_file);
        }
        else {
            bool text;
            s >> text;
            non_executable_file nex_file(x, size, index, sys, hid, name, text);
            add_file(nex_file);
        }
    }
}

void dir::print_to_file(const std::string& name) const {
    std::ofstream f;
    f.open(name);
    std::string str;
    for (int i = 0; i < next_ind; i++) {
        str = get_file_to_str(i);
        f << str;
        if (i != next_ind - 1) {
            f << "\n";
        }
    }
    f.close();
}

bool dir::check_str(const std::string& str) const {
    std::regex re("[A-Za-zA-Яa-я0-9\\s\\!,\\?\\\"\\.:\\;\\']*");
    if (!(std::regex_search(str, re))) {
        return false;
    }
    std::regex re_2("\\s{2,}");
    if (std::regex_search(str, re_2)) {
        return false;
    }
    std::regex re_3("[\\!\\?:\\.\\,\\;]{2,}");
    if (std::regex_search(str, re_3)) {
        return false;
    }
    std::regex re_4("[\\'\\\"]{2,}");
    if (std::regex_search(str, re_4)) {
        return false;
    }
    std::regex re_5("^\"[A-ZА-Я]");
    if (!std::regex_search(str, re_5)) {
        return false;
    }
    return true;
}

void dir::print_all_with_2_or_more_words() const {
    for (int i = 0; i < next_ind; i++) {
        std::regex re(".*.+.");
        std::string str = files[i]->get_name();
        if (std::regex_search(str, re)) {
            std::cout << i + 1 << " " << *(files + i) << "\n";
        }
    }
}

void dir::sort_files(bool (*f)(file&, file&)) {
    bool check = false;
    file* temp;
    do {
        check = false;
        for (int i = 0; i < next_ind - 1; i++) {
            if (f(*(files[i]), *(files[i + 1]))) {
                temp = files[i];
                files[i] = files[i + 1];
            }
        }
    } while (check);
}

```

```

        files[i + 1] = temp;
        check = true;
    }
} while (check);
}
bool sort_version(file& first, file& second) {
    return (first.get_x() < second.get_x());
}
bool sort_size(file& first, file& second) {
    return (first.get_size() < second.get_size());
}
bool sort_ind(file& first, file& second) {
    return (first.get_index() > second.get_index());
}
bool sort_opt(file& first, file& second) {
    if (((second.get_sys() && second.get_hid()) == true) && ((first.get_sys() && first.get_hid())
!= true)) {
        return true;
    }
    else if (((second.get_sys() == false) && (second.get_hid() == true)) && (first.get_hid() !=
true)) {
        return true;
    }
    else if (((second.get_hid() == false) && (first.get_hid() == false)) && ((second.get_sys() ==
true) && (first.get_sys() != true))) {
        return true;
    }
    else {
        return false;
    }
}
bool sort_name(file& first, file& second) {
    return (first.get_name() > second.get_name());
}
std::istream& operator>>(std::istream& is, dir& d) {
    int temp;
    executable_file exetemp;
    non_executable_file nontemp;
    while (is >> temp) {
        if (temp == 1) {
            is >> exetemp;
            if (exetemp.type_of_file != -1) {
                d.add_file(exetemp);
            }
        }
        else {
            is >> nontemp;
            if (nontemp.type_of_file != -1) {
                d.add_file(nontemp);
            }
        }
    }
    return is;
}
std::ostream& operator<<(std::ostream& os, dir& d) {
    for (size_t i = 0; i < d.next_ind; i++) {
        os << *(d[i]) << "\n";
    }
    return os;
}
void count_64::operator()(file f) {
    if (f.get_x() == 64) {
        count++;
    }
}
void count_32::operator()(file f) {
    if (f.get_x() == 32) {
        count++;
    }
}

```

```

    }
}
void count_sys::operator()(file f) {
    if (f.get_sys() == true) {
        count++;
    }
}
void count_hid::operator()(file f) {
    if (f.get_hid() == 64) {
        count++;
    }
}
void print(const file& f) {
    std::cout << f << "\n";
}
void pprint(const std::pair<int, file>& p) {
    std::cout << p.first << " : " << p.second << "\n";
}
void print_64(const file& f) {
    if (f.get_x() == 64) {
        std::cout << f << "\n";
    }
}
void print_32(const file& f) {
    if (f.get_x() == 32) {
        std::cout << f << "\n";
    }
}
void print_sys(const file& f) {
    if (f.get_sys() == true) {
        std::cout << f << "\n";
    }
}
void print_hid(const file& f) {
    if (f.get_hid() == true) {
        std::cout << f << "\n";
    }
}
void pprint_64(const std::pair<int, file>& p) {
    if (p.second.get_x() == 64) {
        std::cout << p.first << " : " << p.second << "\n";
    }
}
void pprint_32(const std::pair<int, file>& p) {
    if (p.second.get_x() == 32) {
        std::cout << p.first << " : " << p.second << "\n";
    }
}
void pprint_sys(const std::pair<int, file>& p) {
    if (p.second.get_sys() == true) {
        std::cout << p.first << " : " << p.second << "\n";
    }
}
void pprint_hid(const std::pair<int, file>& p) {
    if (p.second.get_hid() == true) {
        std::cout << p.first << " : " << p.second << "\n";
    }
}
void ccount_64::operator()(std::pair<int, file> p) {
    if (p.second.get_x() == 64) {
        count++;
    }
}
void ccount_32::operator()(std::pair<int, file> p) {
    if (p.second.get_x() == 32) {
        count++;
    }
}
void ccount_sys::operator()(std::pair<int, file> p) {

```

```

        if (p.second.get_sys() == true) {
            count++;
        }
    }
    void ccount_hid::operator()(std::pair<int, file> p) {
        if (p.second.get_hid() == true) {
            count++;
        }
    }
    bool sort_by::operator()(file& f1, file& f2) {
        if (type == 1) {
            return f1.get_x() > f2.get_x();
        }
        else if (type == 2) {
            return f1.get_sys() > f2.get_sys();
        }
        else if (type == 3) {
            return f1.get_hid() > f2.get_hid();
        }
        else if (type == 4) {
            return f1.get_index() < f2.get_index();
        }
        else if (type == 5) {
            return f1.get_size() > f2.get_size();
        }
        else if (type == 6) {
            return f1.get_name() < f2.get_name();
        }
        else if (type == 7) {
            return f1.get_x() < f2.get_x();
        }
        else if (type == 8) {
            return f1.get_sys() < f2.get_sys();
        }
        else if (type == 9) {
            return f1.get_hid() < f2.get_hid();
        }
        else if (type == 10) {
            return f1.get_index() > f2.get_index();
        }
        else if (type == 11) {
            return f1.get_size() < f2.get_size();
        }
        else if (type == 12) {
            return f1.get_name() > f2.get_name();
        }
    }
}

```

menu.h

```

#pragma once
#include "dir.h"

```

```

void old_menu();
void menu();

```

menu.cpp

```

#include "menu.h"

```

```

void old_menu() {
    setlocale(LC_ALL, "Russian");
    int n = 0, temp_i;
    dir directory;
    std::ifstream f("data.txt");
    std::ofstream d;
    f >> directory;
    f.close();
    int s;
    while (n != 4) {
        std::cout << "Выберите желаемую опцию:" << "\n";
        std::cout << "1 - добавить элемент в список." << "\n";
    }
}

```

```

std::cout << "2 - удалить элемент из списка." << "\n";
std::cout << "3 - показать все элементы списка." << "\n";
std::cout << "4 - завершить работу программы." << "\n";
std::cout << "5 - посчитать количество скрытых системных файлов." << "\n";
std::cout << "6 - прочитать данные из файла. " << "\n";
std::cout << "7 - записать текущий список данных в файл. " << "\n";
std::cout << "8 - найти все элементы в названии которых есть 2 или больше слова. " << "\n";
std::cout << "9 - отсортировать массив. " << "\n";
std::cin >> n;
if (n == 1) {
    directory.add_file_by_str("\File\ 0 123 64 0 0 \0");
    std::cout << "Файл добавлен." << "\n";
}
else if (n == 2) {
    std::cout << "Введите номер удаляемого элемента (нумерация начинается с 1): ";
    std::cin >> temp_i;
    directory.del_file(temp_i - 1);
    std::cout << "Файл удалён. " << "\n";
}
else if (n == 3) {
    std::cout << directory;
}
else if (n == 5) {
    std::cout << "Количество скрытых системных файлов: " << directory.count_system() <<
"\n";
}
else if (n == 6) {
    f.open("data.txt");
    f >> directory;
    f.close();
}
else if (n == 7) {
    d.open("data.txt");
    d << directory;
    d.close();
}
else if (n == 8) {
    directory.print_all_with_2_or_more_words();
}
else if (n == 9) {
    std::cout << "Введите номер признака по которому хотите отсортировать массив: 1 - x, 2
- size, 3 - index, 4 - opt, 5 - name. " << "\n";
    std::cin >> s;
    if (s == 1) {
        directory.sort_files(sort_version);
    }
    else if (s == 2) {
        directory.sort_files(sort_size);
    }
    else if (s == 3) {
        directory.sort_files(sort_ind);
    }
    else if (s == 4) {
        s = 0;
        directory.sort_files(sort_opt);
    }
    else if (s == 5) {
        directory.sort_files(sort_name);
    }
}
}
directory.del_all();
}
void menu() {
    setlocale(LC_ALL, "Russian");
    int n, w;
    std::cout << "Выберите с каким типом контейнера будет работать программа на этот раз: " << "\n";
    std::cout << "1 - vector" << "\n";
    std::cout << "2 - list" << "\n";
}

```



```

std::cout << "3 - map" << "\n";
std::cout << "4 - set" << "\n";
std::cout << "Введите цифру соответствующую необходимому контейнеру: ";
std::cin >> n;
if (n == 1) {
    int q = 0;
    std::vector<file> vect;
    executable_file exetemp;
    non_executable_file nontemp;
    std::ifstream f("data.txt");
    while (f >> n) {
        if (n == 1) {
            f >> exetemp;
            vect.push_back(exetemp);
            vect[q].change_index(q);
        }
        else {
            f >> nontemp;
            vect.push_back(nontemp);
            vect[q].change_index(q);
        }
        q++;
    }
    f.close();
    std::cout << "Данные считаны из файла в вектор.\n";
    while (true) {
        std::cout << "Выберите дальнейшие действия: " << "\n";
        std::cout << " 1 - вывести вектор на экран.\n";
        std::cout << " 2 - считать данные из файла в вектор.\n";
        std::cout << " 3 - добавить элемент в вектор.\n";
        std::cout << " 4 - удалить элемент из вектора по индексу.\n";
        std::cout << " 5 - вывести один элемент вектора по индексу.\n";
        std::cout << " 6 - посчитать количество элементов с заданным параметром.\n";
        std::cout << " 7 - вывести все элементы с заданным параметром.\n";
        std::cout << " 8 - сортировка вектора по заданному параметру.\n";
        std::cout << " 9 - считать данные из файла в новый вектор и объединить 2 вектора.\n";
        std::cout << "10 - завершить работу программы.\n";
        std::cout << "Введите число, что соответствует необходимому действию: ";
        std::cin >> n;
        if (n == 1) {
            std::cout << "Вот данные вектора: \n";
            std::for_each(vect.begin(), vect.end(), print);
        }
        else if (n == 2) {
            vect.clear();
            f.open("data.txt");
            q = 0;
            while (f >> w) {
                if (w == 1) {
                    f >> exetemp;
                    vect.push_back(exetemp);
                    vect[q].change_index(q);
                }
                else {
                    f >> nontemp;
                    vect.push_back(nontemp);
                    vect[q].change_index(q);
                }
                q++;
            }
            std::cout << "Данные считаны из файла в вектор.\n";
        }
        else if (n == 3) {
            non_executable_file non(32, 123, 0, false, false, "text", true);
            vect.push_back(non);
            vect[q].change_index(q);
            q++;
            std::cout << "Элемент добавлен в вектор.\n";
        }
    }
}

```

```

else if (n == 4) {
    std::cout << "Введите индекс удаляемого элемента: ";
    std::cin >> w;
    if (w < q) {
        for (int i = w; i < (q - 1); i++) {
            vect[i] = vect[i + 1];
            vect[i].change_index(i);
        }
        vect.pop_back();
        q--;
        std::cout << "Элемент с данным индексом удалён из вектора.\n";
    }
    else {
        std::cout << "Введите правильный индекс.\n";
    }
}
else if (n == 5) {
    std::cout << "Введите индекс нужного элемента:";
    std::cin >> w;
    if (w < q) {
        std::cout << "Вот данные о элементе с данным индексом:\n";
        std::cout << vect[w] << "\n";
    }
    else {
        std::cout << "Введите правильный индекс.\n";
    }
}
else if (n == 6) {
    std::cout << "Выберите параметр элементы с которым будут посчитаны: \n";
    std::cout << "1 - 64-битные файлы.\n";
    std::cout << "2 - 32-битные файлы.\n";
    std::cout << "3 - системные файлы.\n";
    std::cout << "4 - скрытые файлы.\n";
    std::cout << "Введите значение соответствующее нужному параметру: ";
    std::cin >> n;
    if (n == 1) {
        count_64 c = std::for_each(vect.begin(), vect.end(), count_64());
        std::cout << "Найдено " << c.count << " элементов с заданным параметром.\n";
    }
    else if (n == 2) {
        count_32 c = std::for_each(vect.begin(), vect.end(), count_32());
        std::cout << "Найдено " << c.count << " элементов с заданным параметром.\n";
    }
    else if (n == 3) {
        count_sys c = std::for_each(vect.begin(), vect.end(), count_sys());
        std::cout << "Найдено " << c.count << " элементов с заданным параметром.\n";
    }
    else if (n == 4) {
        count_hid c = std::for_each(vect.begin(), vect.end(), count_hid());
        std::cout << "Найдено " << c.count << " элементов с заданным параметром.\n";
    }
    else {
        std::cout << "Такого параметра нет.\n";
    }
}
else if (n == 7) {
    std::cout << "Выберите параметр элементы с которым будут выведены на экран: \n";
    std::cout << "1 - 64-битные файлы.\n";
    std::cout << "2 - 32-битные файлы.\n";
    std::cout << "3 - системные файлы.\n";
    std::cout << "4 - скрытые файлы.\n";
    std::cout << "Введите значение соответствующее нужному параметру: ";
    std::cin >> n;
    if (n == 1) {
        std::cout << "Вот данные о элементах с заданной характеристикой:\n";
        std::for_each(vect.begin(), vect.end(), print_64);
    }
    else if (n == 2) {
        std::cout << "Вот данные о элементах с заданной характеристикой:\n";
    }
}

```

```

        std::for_each(vect.begin(), vect.end(), print_32);
    }
    else if (n == 3) {
        std::cout << "Вот данные о элементах с заданной характеристикой:\n";
        std::for_each(vect.begin(), vect.end(), print_sys);
    }
    else if (n == 4) {
        std::cout << "Вот данные о элементах с заданной характеристикой:\n";
        std::for_each(vect.begin(), vect.end(), print_hid);
    }
    else {
        std::cout << "Такого параметра нет.\n";
    }
}
else if (n == 8) {
    std::cout << "Выберите параметр по которому элементы будут отсортированы: \n";
    std::cout << "1 - битность системы.\n";
    std::cout << "2 - системность файлов.\n";
    std::cout << "3 - являются ли файлы скрытыми.\n";
    std::cout << "4 - индекс файлов.\n";
    std::cout << "5 - размер файлов.\n";
    std::cout << "6 - название файла.\n";
    std::cout << "Введите значение соответствующее нужному параметру: ";
    std::cin >> n;
    if (n == 1) {
        std::cout << "Выберите какие файлы будут сначала 64-битные (1) или 32-битные
(2): ";

        std::cin >> n;
        if (n == 1) {
            std::sort(vect.begin(), vect.end(), sort_by(1));
            std::cout << "Вектор отсортирован по заданному критерию.\n";
        }
        else if (n == 2) {
            std::sort(vect.begin(), vect.end(), sort_by(7));
            std::cout << "Вектор отсортирован по заданному критерию.\n";
        }
        else {
            std::cout << "Такого варианта нет.\n";
        }
    }
    else if (n == 2) {
        std::cout << "Выберите какие файлы будут сначала системные (1) или не системные
(2): ";

        std::cin >> n;
        if (n == 1) {
            std::sort(vect.begin(), vect.end(), sort_by(2));
            std::cout << "Вектор отсортирован по заданному критерию.\n";
        }
        else if (n == 2) {
            std::sort(vect.begin(), vect.end(), sort_by(8));
            std::cout << "Вектор отсортирован по заданному критерию.\n";
        }
        else {
            std::cout << "Такого варианта нет.\n";
        }
    }
    else if (n == 3) {
        std::cout << "Выберите какие файлы будут сначала скрытые (1) или не скрытые (2):
";

        std::cin >> n;
        if (n == 1) {
            std::sort(vect.begin(), vect.end(), sort_by(3));
            std::cout << "Вектор отсортирован по заданному критерию.\n";
        }
        else if (n == 2) {
            std::sort(vect.begin(), vect.end(), sort_by(9));
            std::cout << "Вектор отсортирован по заданному критерию.\n";
        }
        else {

```

```

        std::cout << "Такого варианта нет.\n";
    }
}
else if (n == 4) {
    std::cout << "Выберите как будут отсортированы файлы по возрастанию индекса (1)
или по убыванию (2): ";
    std::cin >> n;
    if (n == 1) {
        std::sort(vect.begin(), vect.end(), sort_by(4));
        std::cout << "Вектор отсортирован по заданному критерию.\n";
    }
    else if (n == 2) {
        std::sort(vect.begin(), vect.end(), sort_by(10));
        std::cout << "Вектор отсортирован по заданному критерию.\n";
    }
    else {
        std::cout << "Такого варианта нет.\n";
    }
}
else if (n == 5) {
    std::cout << "Выберите как будут отсортированы файлы по убыванию размера (1) или
по возрастанию (2): ";
    std::cin >> n;
    if (n == 1) {
        std::sort(vect.begin(), vect.end(), sort_by(5));
        std::cout << "Вектор отсортирован по заданному критерию.\n";
    }
    else if (n == 2) {
        std::sort(vect.begin(), vect.end(), sort_by(11));
        std::cout << "Вектор отсортирован по заданному критерию.\n";
    }
    else {
        std::cout << "Такого варианта нет.\n";
    }
}
else if (n == 6) {
    std::cout << "Выберите какие файлы будут сначала алфавитный порядок (1) или
обратный алфавитному (2): ";
    std::cin >> n;
    if (n == 1) {
        std::sort(vect.begin(), vect.end(), sort_by(6));
        std::cout << "Вектор отсортирован по заданному критерию.\n";
    }
    else if (n == 2) {
        std::sort(vect.begin(), vect.end(), sort_by(12));
        std::cout << "Вектор отсортирован по заданному критерию.\n";
    }
    else {
        std::cout << "Такого варианта нет.\n";
    }
}
else {
    std::cout << "Такого варианта нет.\n";
}
}
else if (n == 9) {
    std::vector<file> tempv;
    f.open("data.txt");
    int r = 0;
    while (f >> w) {
        if (w == 1) {
            f >> exetemp;
            tempv.push_back(exetemp);
            tempv[r].change_index(r);
        }
        else {
            f >> nontemp;
            tempv.push_back(nontemp);
            tempv[r].change_index(r);
        }
    }
}

```

```

    }
    r++;
}
for (int i = r - 1; !(i < 0); i--) {
    vect.push_back(tempv[i]);
    vect[q].change_index(q);
    q++;
    tempv.pop_back();
}
std::cout << "Данные считаны из файла в вектор. И оба вектора объединены.\n";
}
else {
    break;
}
}
}
else if (n == 2) {
    bool check;
    std::string name;
    std::list<file> lst;
    executable_file exetemp;
    non_executable_file nontemp;
    std::ifstream f("data.txt");
    while (f >> n) {
        if (n == 1) {
            f >> exetemp;
            lst.push_back(exetemp);
        }
        else {
            f >> nontemp;
            lst.push_back(nontemp);
        }
    }
    f.close();
    std::cout << "Данные считаны из файла в список.\n";
    while (true) {
        std::cout << "Выберите дальнейшие действия: " << "\n";
        std::cout << "1 - вывести список на экран.\n";
        std::cout << "2 - считать данные из файла в список.\n";
        std::cout << "3 - добавить элемент в список.\n";
        std::cout << "4 - удалить элемент из списка по его названию.\n";
        std::cout << "5 - получить элемент из списка по его названию.\n";
        std::cout << "6 - посчитать количество элементов с заданным параметром.\n";
        std::cout << "7 - вывести все элементы с заданным параметром.\n";
        std::cout << "8 - отсортировать список по заданному параметру.\n";
        std::cout << "9 - завершить работу программы.\n";
        std::cout << "Введите число, что соответствует необходимому действию: ";
        std::cin >> n;
        if (n == 1) {
            std::cout << "Вот данные списка: \n";
            std::for_each(lst.begin(), lst.end(), print);
        }
        else if (n == 2) {
            lst.clear();
            f.open("data.txt");
            while (f >> n) {
                if (n == 1) {
                    f >> exetemp;
                    lst.push_back(exetemp);
                }
                else {
                    f >> nontemp;
                    lst.push_back(nontemp);
                }
            }
            f.close();
            std::cout << "Данные считаны из файла в список.\n";
        }
        else if (n == 3) {

```

```

        non_executable_file non(32, 123, 0, false, false, "text", true);
        lst.push_back(non);
        std::cout << "Элемент добавлен в список.\n";
    }
    else if (n == 4) {
        std::cout << "Введите название удаляемого элемента: ";
        std::cin.ignore();
        std::getline(std::cin, name);
        check = false;
        for (auto el : lst) {
            if (el.get_name() == name) {
                lst.remove(el);
                check = true;
                break;
            }
        }
        if (check) {
            std::cout << "Элемент удалён.\n";
        }
        else {
            std::cout << "Такого элемента нет.\n";
        }
    }
}
else if (n == 5) {
    std::cout << "Введите название нужного элемента: ";
    std::cin.ignore();
    std::getline(std::cin, name);
    check = true;
    for (auto el : lst) {
        if (el.get_name() == name) {
            std::cout << "Вот нужный элемент: ";
            std::cout << el << "\n";
            check = false;
            break;
        }
    }
    if (check) {
        std::cout << "Такого элемента нет.\n";
    }
}
else if (n == 6) {
    std::cout << "Выберите параметр элементы с которым будут посчитаны: \n";
    std::cout << "1 - 64-битные файлы.\n";
    std::cout << "2 - 32-битные файлы.\n";
    std::cout << "3 - системные файлы.\n";
    std::cout << "4 - скрытые файлы.\n";
    std::cout << "Введите значение соответствующее нужному параметру: ";
    std::cin >> n;
    if (n == 1) {
        count_64 c = std::for_each(lst.begin(), lst.end(), count_64());
        std::cout << "Найдено " << c.count << " элементов с заданным параметром.\n";
    }
    else if (n == 2) {
        count_32 c = std::for_each(lst.begin(), lst.end(), count_32());
        std::cout << "Найдено " << c.count << " элементов с заданным параметром.\n";
    }
    else if (n == 3) {
        count_sys c = std::for_each(lst.begin(), lst.end(), count_sys());
        std::cout << "Найдено " << c.count << " элементов с заданным параметром.\n";
    }
    else if (n == 4) {
        count_hid c = std::for_each(lst.begin(), lst.end(), count_hid());
        std::cout << "Найдено " << c.count << " элементов с заданным параметром.\n";
    }
    else {
        std::cout << "Такого параметра нет.\n";
    }
}
else if (n == 7) {

```

```

std::cout << "Выберите параметр элементы с которым будут выведены на экран: \n";
std::cout << "1 - 64-битные файлы.\n";
std::cout << "2 - 32-битные файлы.\n";
std::cout << "3 - системные файлы.\n";
std::cout << "4 - скрытые файлы.\n";
std::cout << "Введите значение соответствующее нужному параметру: ";
std::cin >> n;
if (n == 1) {
    std::cout << "Вот данные о элементах с заданной характеристикой:\n";
    std::for_each(lst.begin(), lst.end(), print_64);
}
else if (n == 2) {
    std::cout << "Вот данные о элементах с заданной характеристикой:\n";
    std::for_each(lst.begin(), lst.end(), print_32);
}
else if (n == 3) {
    std::cout << "Вот данные о элементах с заданной характеристикой:\n";
    std::for_each(lst.begin(), lst.end(), print_sys);
}
else if (n == 4) {
    std::cout << "Вот данные о элементах с заданной характеристикой:\n";
    std::for_each(lst.begin(), lst.end(), print_hid);
}
else {
    std::cout << "Такого параметра нет.\n";
}
}
else if (n == 8) {
    std::cout << "Выберите параметр по которому элементы будут отсортированы: \n";
    std::cout << "1 - битность системы.\n";
    std::cout << "2 - системность файлов.\n";
    std::cout << "3 - являются ли файлы скрытыми.\n";
    std::cout << "4 - индекс файлов.\n";
    std::cout << "5 - размер файлов.\n";
    std::cout << "6 - название файла.\n";
    std::cout << "Введите значение соответствующее нужному параметру: ";
    std::cin >> n;
    if (n == 1) {
        std::cout << "Выберите какие файлы будут сначала 64-битные (1) или 32-битные
(2): ";

        std::cin >> n;
        if (n == 1) {
            lst.sort(sort_by(1));
            std::cout << "Список отсортирован по заданному критерию.\n";
        }
        else if (n == 2) {
            lst.sort(sort_by(7));
            std::cout << "Список отсортирован по заданному критерию.\n";
        }
        else {
            std::cout << "Такого варианта нет.\n";
        }
    }
    else if (n == 2) {
        std::cout << "Выберите какие файлы будут сначала системные (1) или не системные
(2): ";

        std::cin >> n;
        if (n == 1) {
            lst.sort(sort_by(2));
            std::cout << "Список отсортирован по заданному критерию.\n";
        }
        else if (n == 2) {
            lst.sort(sort_by(8));
            std::cout << "Список отсортирован по заданному критерию.\n";
        }
        else {
            std::cout << "Такого варианта нет.\n";
        }
    }
}
}

```



```
else if (n == 3) {
    std::cout << "Выберите какие файлы будут сначала скрытые (1) или не скрытые (2): ";

    std::cin >> n;
    if (n == 1) {
        lst.sort(sort_by(3));
        std::cout << "Список отсортирован по заданному критерию.\n";
    }
    else if (n == 2) {
        lst.sort(sort_by(9));
        std::cout << "Список отсортирован по заданному критерию.\n";
    }
    else {
        std::cout << "Такого варианта нет.\n";
    }
}
else if (n == 4) {
    std::cout << "Выберите какие файлы будут сначала больший индекс (1) или меньший индекс (2): ";

    std::cin >> n;
    if (n == 1) {
        lst.sort(sort_by(4));
        std::cout << "Список отсортирован по заданному критерию.\n";
    }
    else if (n == 2) {
        lst.sort(sort_by(10));
        std::cout << "Список отсортирован по заданному критерию.\n";
    }
    else {
        std::cout << "Такого варианта нет.\n";
    }
}
else if (n == 5) {
    std::cout << "Выберите какие файлы будут сначала больший файл (1) или меньший файл (2): ";

    std::cin >> n;
    if (n == 1) {
        lst.sort(sort_by(5));
        std::cout << "Список отсортирован по заданному критерию.\n";
    }
    else if (n == 2) {
        lst.sort(sort_by(11));
        std::cout << "Список отсортирован по заданному критерию.\n";
    }
    else {
        std::cout << "Такого варианта нет.\n";
    }
}
else if (n == 6) {
    std::cout << "Выберите какие файлы будут сначала алфавитный порядок (1) или обратный алфавитному (2): ";
    std::cin >> n;
    if (n == 1) {
        lst.sort(sort_by(6));
        std::cout << "Список отсортирован по заданному критерию.\n";
    }
    else if (n == 2) {
        lst.sort(sort_by(12));
        std::cout << "Список отсортирован по заданному критерию.\n";
    }
    else {
        std::cout << "Такого варианта нет.\n";
    }
}
else {
    std::cout << "Такого варианта нет.\n";
}
```

```

        break;
    }
}
}
else if (n == 3) {
    int q = 0;
    bool check;
    std::multimap<int, file> mp;
    executable_file exetemp;
    non_executable_file nontemp;
    std::ifstream f("data.txt");
    while (f >> n) {
        if (n == 1) {
            f >> exetemp;
            mp.insert(std::pair<int, file>(q++, exetemp));
        }
        else {
            f >> nontemp;
            mp.insert(std::pair<int, file>(q++, nontemp));
        }
    }
    f.close();
    std::cout << "Данные считаны из файла в контейнер.\n";
    while (true) {
        std::cout << "Выберите дальнейшие действия: " << "\n";
        std::cout << "1 - вывести содержимое контейнера на экран.\n";
        std::cout << "2 - считать данные из файла в контейнер.\n";
        std::cout << "3 - добавить элемент в контейнер.\n";
        std::cout << "4 - удалить элемент из контейнера по его ключу.\n";
        std::cout << "5 - получить элемент из контейнера по его ключу.\n";
        std::cout << "6 - посчитать количество элементов с заданным параметром.\n";
        std::cout << "7 - вывести все элементы с заданным параметром.\n";
        std::cout << "8 - считать данные из файла в новый контейнер и объединить контейнеры.\n";
        std::cout << "9 - завершить работу программы.\n";
        std::cout << "Введите число, что соответствует необходимому действию: ";
        std::cin >> n;
        if (n == 1) {
            std::cout << "Вот данные вашего контейнера: \n";
            std::for_each(mp.begin(), mp.end(), pprint);
        }
        else if (n == 2) {
            mp.clear();
            f.open("data.txt");
            while (f >> n) {
                if (n == 1) {
                    f >> exetemp;
                    mp.insert(std::pair<int, file>(q++, exetemp));
                }
                else {
                    f >> nontemp;
                    mp.insert(std::pair<int, file>(q++, nontemp));
                }
            }
            f.close();
            std::cout << "Данные считаны из файла в контейнер.\n";
        }
        else if (n == 3) {
            non_executable_file non(32, 123, 0, false, false, "text", true);
            mp.insert(std::pair<int, file>(q++, non));
            std::cout << "Элемент добавлен в контейнер.\n";
        }
        else if (n == 4) {
            std::cout << "Введите числовой ключ удаляемого элемента: ";
            std::cin >> w;
            check = false;
            for (auto el : mp) {
                if (el.first == w) {
                    mp.erase(w);
                    check = true;
                }
            }
            if (!check) {
                std::cout << "Элемент с таким ключом не найден.\n";
            }
        }
    }
}

```

```

        break;
    }
}
if (check) {
    std::cout << "Элемент удалён из контейнера.\n";
}
else {
    std::cout << "Неверный ключ.\n";
}
}
else if (n == 5) {
    std::cout << "Введите числовой ключ нужного элемента: ";
    std::cin >> w;
    check = true;
    for (auto el : mp) {
        if (el.first == w) {
            std::cout << "Вот данные нужного элемента: " << el.second << "\n";
            check = false;
            break;
        }
    }
    if (check) {
        std::cout << "Неверный ключ.\n";
    }
}
else if (n == 6) {
    std::cout << "Выберите параметр элементы с которым будут посчитаны: \n";
    std::cout << "1 - 64-битные файлы.\n";
    std::cout << "2 - 32-битные файлы.\n";
    std::cout << "3 - системные файлы.\n";
    std::cout << "4 - скрытые файлы.\n";
    std::cout << "Введите значение соответствующее нужному параметру: ";
    std::cin >> n;
    if (n == 1) {
        ccount_64 c = std::for_each(mp.begin(), mp.end(), ccount_64());
        std::cout << "Найдено " << c.count << " элементов с заданным параметром.\n";
    }
    else if (n == 2) {
        ccount_32 c = std::for_each(mp.begin(), mp.end(), ccount_32());
        std::cout << "Найдено " << c.count << " элементов с заданным параметром.\n";
    }
    else if (n == 3) {
        ccount_sys c = std::for_each(mp.begin(), mp.end(), ccount_sys());
        std::cout << "Найдено " << c.count << " элементов с заданным параметром.\n";
    }
    else if (n == 4) {
        ccount_hid c = std::for_each(mp.begin(), mp.end(), ccount_hid());
        std::cout << "Найдено " << c.count << " элементов с заданным параметром.\n";
    }
    else {
        std::cout << "Такого параметра нет.\n";
    }
}
else if (n == 7) {
    std::cout << "Выберите параметр элементы с которым будут выведены на экран: \n";
    std::cout << "1 - 64-битные файлы.\n";
    std::cout << "2 - 32-битные файлы.\n";
    std::cout << "3 - системные файлы.\n";
    std::cout << "4 - скрытые файлы.\n";
    std::cout << "Введите значение соответствующее нужному параметру: ";
    std::cin >> n;
    if (n == 1) {
        std::cout << "Вот данные о элементах с заданной характеристикой:\n";
        std::for_each(mp.begin(), mp.end(), pprint_64);
    }
    else if (n == 2) {
        std::cout << "Вот данные о элементах с заданной характеристикой:\n";
        std::for_each(mp.begin(), mp.end(), pprint_32);
    }
}

```

```

else if (n == 3) {
    std::cout << "Вот данные о элементах с заданной характеристикой:\n";
    std::for_each(mp.begin(), mp.end(), pprint_sys);
}
else if (n == 4) {
    std::cout << "Вот данные о элементах с заданной характеристикой:\n";
    std::for_each(mp.begin(), mp.end(), pprint_hid);
}
else {
    std::cout << "Такого параметра нет.\n";
}
}
else if (n == 8) {
    std::multimap<int, file> tempm;
    int k = 0;
    f.open("data.txt");
    while (f >> n) {
        if (n == 1) {
            f >> exetemp;
            tempm.insert(std::pair<int, file>(k++, exetemp));
        }
        else {
            f >> nontemp;
            tempm.insert(std::pair<int, file>(k++, nontemp));
        }
    }
    f.close();
    for (auto el : tempm) {
        check = true;
        for (auto le : mp) {
            if (el == le) {
                check = false;
            }
        }
        if (check) {
            mp.insert(el);
        }
    }
    std::cout << "Данные считаны из файла в дополнительный контейнер. Затем, основной и
дополнительный контейнеры объединены.\n";
}
else {
    break;
}
}
else if (n == 4) {
    bool check;
    std::string name;
    std::set<file> st;
    executable_file exetemp;
    non_executable_file nontemp;
    std::ifstream f("data.txt");
    while (f >> n) {
        if (n == 1) {
            f >> exetemp;
            st.insert(exetemp);
        }
        else {
            f >> nontemp;
            st.insert(nontemp);
        }
    }
    f.close();
    std::cout << "Данные считаны из файла в множество.\n";
    while (true) {
        std::cout << "Выберите дальнейшие действия: " << "\n";
        std::cout << "1 - вывести содержимое множества на экран.\n";
        std::cout << "2 - считать данные из файла в множество.\n";
    }
}

```

```

std::cout << "3 - добавить элемент в множество.\n";
std::cout << "4 - удалить элемент из множества по названию.\n";
std::cout << "5 - получить элемент из множества по названию.\n";
std::cout << "6 - посчитать количество элементов с заданным параметром.\n";
std::cout << "7 - вывести все элементы с заданным параметром.\n";
std::cout << "8 - завершить работу программы.\n";
std::cout << "Введите число, что соответствует необходимому действию: ";
std::cin >> n;
if (n == 1) {
    std::cout << "Вот данные из вашего множества: \n";
    std::for_each(st.begin(), st.end(), print);
}
else if (n == 2) {
    st.clear();
    f.open("data.txt");
    while (f >> n) {
        if (n == 1) {
            f >> exetemp;
            st.insert(exetemp);
        }
        else {
            f >> nontemp;
            st.insert(nontemp);
        }
    }
    f.close();
    std::cout << "Данные считаны из файла в множество.\n";
}
else if (n == 3) {
    non_executable_file non(32, 123, 0, false, false, "text", true);
    st.insert(non);
    std::cout << "Элемент добавлен в множество.\n";
}
else if (n == 4) {
    std::cout << "Введите название удаляемого элемента: ";
    std::cin.ignore();
    std::getline(std::cin, name);
    check = false;
    for (auto el : st) {
        if (el.get_name() == name) {
            st.erase(el);
            check = true;;
            break;
        }
    }
    if (check) {
        std::cout << "Элемент был удалён из множества.\n";
    }
    else {
        std::cout << "Элемента с таким названием в множестве нет.\n";
    }
}
else if (n == 5) {
    std::cout << "Введите название нужного элемента: ";
    std::cin.ignore();
    std::getline(std::cin, name);
    check = true;
    for (auto el : st) {
        if (el.get_name() == name) {
            std::cout << "Вот ваш элемент: ";
            std::cout << el << "\n";
            check = false;
            break;
        }
    }
    if (check) {
        std::cout << "Элемента с таким названием в множестве нет.\n";
    }
}
}

```

```

else if (n == 6) {
    std::cout << "Выберите параметр элементы с которым будут посчитаны: \n";
    std::cout << "1 - 64-битные файлы.\n";
    std::cout << "2 - 32-битные файлы.\n";
    std::cout << "3 - системные файлы.\n";
    std::cout << "4 - скрытые файлы.\n";
    std::cout << "Введите значение соответствующее нужному параметру: ";
    std::cin >> n;
    if (n == 1) {
        count_64 c = std::for_each(st.begin(), st.end(), count_64());
        std::cout << "Найдено " << c.count << " элементов с заданным параметром.\n";
    }
    else if (n == 2) {
        count_32 c = std::for_each(st.begin(), st.end(), count_32());
        std::cout << "Найдено " << c.count << " элементов с заданным параметром.\n";
    }
    else if (n == 3) {
        count_sys c = std::for_each(st.begin(), st.end(), count_sys());
        std::cout << "Найдено " << c.count << " элементов с заданным параметром.\n";
    }
    else if (n == 4) {
        count_hid c = std::for_each(st.begin(), st.end(), count_hid());
        std::cout << "Найдено " << c.count << " элементов с заданным параметром.\n";
    }
    else {
        std::cout << "Такого параметра нет.\n";
    }
}
else if (n == 7) {
    std::cout << "Выберите параметр элементы с которым будут выведены на экран: \n";
    std::cout << "1 - 64-битные файлы.\n";
    std::cout << "2 - 32-битные файлы.\n";
    std::cout << "3 - системные файлы.\n";
    std::cout << "4 - скрытые файлы.\n";
    std::cout << "Введите значение соответствующее нужному параметру: ";
    std::cin >> n;
    if (n == 1) {
        std::cout << "Вот данные о элементах с заданной характеристикой:\n";
        std::for_each(st.begin(), st.end(), print_64);
    }
    else if (n == 2) {
        std::cout << "Вот данные о элементах с заданной характеристикой:\n";
        std::for_each(st.begin(), st.end(), print_32);
    }
    else if (n == 3) {
        std::cout << "Вот данные о элементах с заданной характеристикой:\n";
        std::for_each(st.begin(), st.end(), print_sys);
    }
    else if (n == 4) {
        std::cout << "Вот данные о элементах с заданной характеристикой:\n";
        std::for_each(st.begin(), st.end(), print_hid);
    }
    else {
        std::cout << "Такого параметра нет.\n";
    }
}
else {
    break;
}
}
}

tests.cpp
#include "menu.h"
#define _CRTDBG_MAP_ALLOC

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

```

```

executable_file exe(64, 321, 5, true, true, "abc", 123);
non_executable_file non(32, 123, 0, false, false, "text", true);
sort_by s1(1);
sort_by s2(2);
sort_by s3(3);
sort_by s4(4);
sort_by s5(5);
sort_by s6(6);
sort_by s7(7);
sort_by s8(8);
sort_by s9(9);
sort_by s10(10);
sort_by s11(11);
sort_by s12(12);
std::cout << "Тест на работу функтора будет пройден если на следующей строке будет выведен
такой текст: 1 1 1 0 1 1 0 0 1 0 0 0\n" << s1(exe, non) << " " << s2(exe, non) << " " << s3(exe,
non) << " " << s4(exe, non) << " " << s5(exe, non) << " " << s6(exe, non) << " " << s7(exe, non) <<
" " << s8(exe, non) << " " << s9(exe, non) << " " << s10(exe, non) << " " << s11(exe, non) << " " <<
s12(exe, non) << "\n";
if (_CrtDumpMemoryLeaks()) {
    std::cout << "Утечка памяти обнаружена." << "\n";
}
else {
    std::cout << "Утечка памяти не обнаружена." << "\n";
}
int n;
std::cin >> n;
}

```

data.txt

```

1 "Sys file" 2 123 32 1 1 100
2 "Qwerty" 3 521 64 1 0 1
1 "Hello friend" 4 289 64 0 1 20
2 "Water" 5 10000 32 0 0 0

```

4. Результаты работы программы

Результаты работы программы:

```

Выберите с каким типом контейнера будет работать программа на этот раз:
1 - vector
2 - list
3 - map
4 - set
Введите цифру соответствующую необходимому контейнеру: 1
Файл создан при помощи конструктора по умолчанию.
Файл создан при помощи конструктора по умолчанию.
Файл создан при помощи конструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Файл создан при помощи конструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Файл создан при помощи конструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Файл создан при помощи конструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Данные считаны из файла в вектор.
Выберите дальнейшие действия:
1 - вывести вектор на экран.
2 - считать данные из файла в вектор.
3 - добавить элемент в вектор.
4 - удалить элемент из вектора по индексу.
5 - вывести один элемент вектора по индексу.
6 - посчитать количество элементов с заданным параметром.
7 - вывести все элементы с заданным параметром.
8 - сортировка вектора по заданному параметру.
9 - считать данные из файла в новый вектор и объединить 2 вектора.
10 - завершить работу программы.
Введите число, что соответствует необходимому действию:

```


Результати тестів:

```
Файл создан при помощи конструктора с аргументами.  
Файл создан при помощи конструктора с аргументами.  
Тест на работу функтора будет пройден если на следующей строке будет выведен такой текст: 1 1 1 0 1 1 0 0 0 1 0 0  
1 1 1 0 1 1 0 0 0 1 0 0  
Утечка памяти не обнаружена.
```

5. Висновки

При виконанні даної лабораторної роботи було створено функтори для сортування об'єктів класу файл у стандартних контейнерах, які було використано в функціях sort, яка в випадку з списком була оформлена як метод. Окрім цього була додана можливість створити нові контейнери типів вектор і map, з послідовним їх злиттям з початковим контейнером.

Програма протестована, витоків пам'яті немає, виконується без помилок.