

Звіт

Автор: Капелька Я.І. КІТ-119а Дата: 29 травня 2020

Лабораторна робота №13. АЛГОРИТМИ ПЕРЕМІЩЕННЯ ТА ПОШУКУ

Тема. STL. Алгоритми переміщення та пошуку.

Мета: на практиці порівняти STL-алгоритми, що не модифікують послідовність.

1. Завдання до роботи **Індивідуальне**

завдання:

Поширити попередню лабораторну роботу додавши функції пошуку елемента за заданими параметрами і підрахування кількості елементів з заданим параметром.

2. Опис класів, змінних, методів та функцій

2.1 Опис класів

Базовий клас: CCountry

Клас нащадок базового класу: Inheritor_CCountry та Inheritor_CCountry_second

Клас, що має в собі масив базового класу та методи для роботи з ним: CMethod й його аналог для класу нащадку Inheritor_CMethod

Клас, що повинен демонструвати композицію: CCitizen

2.2 Опис змінних

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

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

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

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

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

`std::string place_of_birth_citizen` – поле класу `CCitizen`(місце народження жителя міста).

`Cint birthday_citizen` – поле класу `CCitizen`(дата народження жителя міста).

`Cint number_of_cities` – поле класу `CCountry`(кількість міст.).

`Cint population` – поле класу `CCountry`(популяція).

`Cint area` – поле класу `CCountry`(площа).

`Cint unical_index` – поле класу `CCountry`(унікальний індекс).

`Cint population_density` – поле класу `CCountry`(щільність населення).

`std::string title` – поле класу `CCountry`(назва країни).

`CCitizen citizen` – поле класу `CCountry`(місце і дата народження жителя міста).

`Cint next_i` – поле класу `CMetod`(номер наступного файлу у директорії).

`Cint new_i` – поле класу `CMetod`(індекс наступного файлу у директорії).

`CCountry** countries` – поле класу `CMetod`(масив елементів класу `CCountry`).

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

`bool monarchy` – поле класу `Inheritor_CCountry` (чи встановлена в країні монархія).

`bool gross_domestic_product` – поле класу `Inheritor_CCountry_second` (чи є ВВП в країні).

2.3 Опис методів

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

`virtual Cint getPopulation () const` – отримання значення поля `population` змінної класу `CCountry`(метод класу `CCountry`).

`virtual Cint getArea () const` – отримання значення поля `area` змінної класу `CCountry`(метод класу `CCountry`).

`virtual Cint getUnical_index () const` – отримання значення поля `unical_index` змінної класу `CCountry`(метод класу `CCountry`).

`virtual Cint` getPopulation_density () `const` – отримання значення поля `population_density` змінної класу `CCountry`(метод класу `CCountry`).

`virtual std::string` getTitle() `const` – отримання значення поля `title` змінної класу `CCountry`(метод класу `CCountry`).

`virtual void` setNumber_of_cities (`const int` &Number_of_cities) – зміна значення поля `number_of_cities` змінної класу `CCountry`(метод класу `CCountry`).

`virtual void` setPopulation (`const int` &Population) – зміна значення поля `population` змінної класу `CCountry`(метод класу `CCountry`).

`virtual void` setArea (`const int` &Area) – зміна значення поля `area` змінної класу `CCountry`(метод класу `CCountry`).

`virtual void` setUnical_index (`const int`& Unical_index) – зміна значення поля `unical_index` змінної класу `CCountry`(метод класу `CCountry`).

`virtual void` setPopulation_density (`const int`& Population_density) – зміна значення поля `population_density` змінної класу `CCountry`(метод класу `CCountry`).

`virtual void` setTitle(`const std::string`& Title) – зміна значення поля `title` змінної класу `CCountry`(метод класу `CCountry`).

`const std::string` getPlace_of_birth_citizen() `const` – отримання значення поля `place_of_birth_citizen` змінної класу `CCountry`(метод класу `CCountry`).

`Cint` getBirthday_citizen() `const` – отримання значення поля `birthday_citizen` змінної класу `CCountry`(метод класу `CCountry`).

`void` setPlace_of_birth_citizen(`const std::string`& Place_of_birth_citizen) – зміна значення поля `place_of_birth_citizen` змінної класу `CCountry`(метод класу `CCountry`).

`void` setBirthday_citizen(`const int`& Birthday_citizen) – зміна значення поля `birthday_citizen` змінної класу `CCountry`(метод класу `CCountry`).

`CCountry()` – конструктор класу `CCountry`.

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

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

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

`void` add_el(`const CCountry` & CCountry) – додавання об'єкту класу `CCountry` до масиву в класі `CMetod`(метод класу `CMetod`).

`void` remove_el(`const int` &index) – видалення об'єкту класу `CCountry` з масиву в класі `CMetod`(метод класу `CMetod`).

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

`void find_to_str_by_file (const std::string& str)` – додавання об'єкту класу `CCountry` до масиву в класі `CMetod` за допомогою строки з інформацією про об'єкт(метод класу `CMetod`).

`void read_from_file(const std::string& name)` – заповнення масиву об'єктів класу `CCountry` інформація про які буде зчитана з файлу(метод класу `CMetod`).

`CCountry find_to_index(const int& index) const` – отримання об'єкту класу `CCountry` з масиву в класі `CMetod`(метод класу `CMetod`).

`void get_to_Screen(const int &index) const` – виведення об'єкту класу `CCountry` з масиву в класі `CMetod` на екран(метод класу `CMetod`).

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

`void find_to_population_density() const` – визначення, яка країна має найменшу щільність населення в об'єкті класу `CMetod`(метод класу `CMetod`).

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

`void get_str_by_file (const int &index) const` – запис у рядок інформації про об'єкт класу `CCountry` (метод класу `CMetod`).

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

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

`void sort(*comp)(CCountry&,CCountry&))` – сортування усіх об'єктів класу `CCountry` в об'єкті класу `CMetod` на екран(метод класу `CMetod`).

`virtual bool getMonarchy() const` – метод класу `CCountry` перевантажений у класі `Inhebitor_CCountry`.

`Inheritor_CCountry()` – конструктор класу `Inheritor_CCountry`.

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

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

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

`Inheritor_CCountry_second ()` – конструктор класу `Inheritor_CCountry_second`.

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

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

~ Inheritor_CCountry_second() – деструктор класу

Inheritor_CCountry_second.

virtual std::string getInfo() 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 bool getMonarchy() const override final – отримання значення поля monarchy змінної класу Inheritor_CCountry (метод класу Inheritor_CCountry).

virtual void setMonarchy(const bool&) final – зміна значення поля monarchy змінної класу Inheritor_CCountry (метод класу Inheritor_CCountry).

virtual bool getGross_domestic_product () const final – метод класу Inheritor_CCountry_second, повертає значення поля gross_domestic_product.

virtual void setGross_domestic_product (const bool&) final – метод класу Inheritor_CCountry_second, змінює значення поля gross_domestic_product.

2.4 Опис функцій

void menu() – функція меню.

void old_menu() – функція меню.

bool sortTitle(CCountry&, CCountry&) – функція порівняння двох країн по їх назві.

bool sortNumber_of_cities(CCountry&, CCountry&) – функція порівняння двох країн по їх кількість міст.

bool sortPopulation(CCountry&, CCountry&) – функція порівняння двох країн по їх популяції.

bool sortArea(CCountry&, CCountry&) – функція порівняння двох країн по їх площі.

bool sortCitizen(CCountry&, CCountry&) – функція порівняння двох країн по їх місцю та дню народження жителя міста.

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

bool operator==(const CCountry& Country1, const CCountry& Country2) – перевантаження оператора порівняння.

bool operator!=(const CCountry & Country1, const CCountry & Country2) – перевантаження ще одного оператора порівняння.

bool operator==(const Inheritor_CCountry& Inheritor_Country1, const Inheritor_CCountry & Inheritor_Country2) – аналогічне перевантаження для класу нащадку.

bool operator!=(const Inheritor_CCountry & Inheritor_Country1, const Inheritor_CCountry & Inheritor_Country2) – аналогічне перевантаження для класу нащадку.

bool operator==(const Inheritor_CCountry_second & Inheritor_Country_second1, const Inheritor_CCountry_second &

Inheritor_Country_second2) – аналогічне перевантаження для класу нащадку.

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

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

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

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

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

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

3 Текст програми

Лабораторная работа 13.cpp

```
#include "menu.h"
int main()
{
    menu();
}
CCountry.h
#pragma once
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <string>
#include <fstream>
#include <sstream>
#include <regex>
#include <iomanip>
#include <vector>
#include <set>
#include <list>
#include <map>
typedef int Cint;
class CCitizen
{
private:
    bool place_of_birth_citizen;
    bool birthday_citizen;
public:
    const bool getPlace_of_birth_citizen() const;
    const bool getBirthday_citizen() const;
    void setPlace_of_birth_citizen(const bool&);
    void setBirthday_citizen(const bool&);
};
class CCountry
{
protected:
    std::string title;
    Cint population_density;
    Cint number_of_cities;
    Cint population;
    Cint area;
    Cint unical_index;
    CCitizen citizen;
public:
```

```

    Cint type_of_Country = 0;
    CCountry();
    CCountry(const CCountry&);
    CCountry(const std::string&, const int&, const int&, const int&, const bool&, const
bool&, const int&);
    ~CCountry();
    virtual bool getPlace_of_birth_citizen() const;
    virtual bool getBirthday_citizen() const;
    virtual std::string getTitle() const;
    virtual Cint getPopulation_density() const;
    virtual Cint getNumber_of_cities() const;
    virtual Cint getPopulation() const;
    virtual Cint getArea() const;
    virtual Cint getUnical_index() const;
    virtual void setTitle(const std::string&);
    virtual void setPopulation_density(const int&);
    virtual void setNumber_of_cities(const int&);
    virtual void setPopulation(const int&);
    virtual void setArea(const int&);
    virtual void setUnical_index(const int&);
    virtual void setPlace_of_birth_citizen(const bool&);
    virtual void setBirthday_citizen(const bool&);
    virtual std::string getInfo() const;
    CCountry& operator= (const CCountry& Country);
    friend bool operator== (const CCountry& Country1, const CCountry& Country2);
    friend bool operator!= (const CCountry& Country1, const CCountry& Country2);
};
class Inheritor_CCountry final : public CCountry
{
private:
    bool monarchy;
public:
    virtual bool getMonarchy() const final;
    virtual void setMonarchy(const bool&) final;
    Inheritor_CCountry();
    Inheritor_CCountry(const Inheritor_CCountry&);
    Inheritor_CCountry(const std::string&, const int&, const int&, const int&, const
bool&, const bool&, const int&, const bool&);
    ~Inheritor_CCountry();
    virtual std::string getInfo() const final;
    Inheritor_CCountry& operator= (const Inheritor_CCountry& Inheritor_Country);
    friend bool operator== (const Inheritor_CCountry& Inheritor_Country1, const
Inheritor_CCountry& Inheritor_Country2);
    friend bool operator!= (const Inheritor_CCountry& Inheritor_Country1, const
Inheritor_CCountry& Inheritor_Country2);
};
class Inheritor_CCountry_second final : public CCountry
{
private:
    bool gross_domestic_product;
public:
    virtual bool getGross_domestic_product() const final;
    virtual void setGross_domestic_product(const bool&) final;
    Inheritor_CCountry_second();
    Inheritor_CCountry_second(const Inheritor_CCountry_second&);
    Inheritor_CCountry_second(const std::string&, const int&, const int&, const int&,
const bool&, const bool&, const int&, const bool&);
    ~Inheritor_CCountry_second();
    virtual std::string getInfo() const final;
    Inheritor_CCountry_second& operator=(const Inheritor_CCountry_second&
Inheritor_Country_second);
    friend bool operator== (const Inheritor_CCountry_second& Inheritor_Country_second1,
const Inheritor_CCountry_second& Inheritor_Country_second2);
    friend bool operator!= (const Inheritor_CCountry_second& Inheritor_Country_second1,
const Inheritor_CCountry_second& Inheritor_Country_second2);
};
bool check_str(const std::string& str);

```



```

bool operator== (const CCountry& Country1, const CCountry& Country2);
bool operator!= (const CCountry& Country1, const CCountry& Country2);
bool operator== (const Inheritor_CCountry& Inheritor_Country1, const Inheritor_CCountry&
Inheritor_Country2);
bool operator!= (const Inheritor_CCountry& Inheritor_Country1, const Inheritor_CCountry&
Inheritor_Country2);
bool operator== (const Inheritor_CCountry_second& Inheritor_Country_second1, const
Inheritor_CCountry_second& Inheritor_Country_second2);
bool operator!= (const Inheritor_CCountry_second& Inheritor_Country_second1, const
Inheritor_CCountry_second& Inheritor_Country_second2);
bool operator> (const CCountry& Country1, const CCountry& Country2);
bool operator< (const CCountry& Country1, const CCountry& Country2);
bool operator> (const Inheritor_CCountry& Inheritor_Country1, const Inheritor_CCountry&
Inheritor_Country2);
bool operator< (const Inheritor_CCountry& Inheritor_Country1, const Inheritor_CCountry&
Inheritor_Country2);
bool operator> (const Inheritor_CCountry_second& Inheritor_Country_second1, const
Inheritor_CCountry_second& Inheritor_Country_second2);
bool operator< (const Inheritor_CCountry_second& Inheritor_Country_second1, const
Inheritor_CCountry_second& Inheritor_Country_second2);
std::ostream& operator<< (std::ostream& os, const Inheritor_CCountry& Inheritor_Country);
std::ostream& operator<< (std::ostream& os, const Inheritor_CCountry_second&
Inheritor_Country_second);
std::istream& operator>> (std::istream& is, Inheritor_CCountry& Inheritor_Country);
std::istream& operator>> (std::istream& is, Inheritor_CCountry_second&
Inheritor_Country_second);
std::ostream& operator<< (std::ostream& os, const CCountry& Country);
CCountry.cpp
#include "CCountry.h"
std::string CCountry::getTitle() const { return title; }
Cint CCountry::getPopulation_density() const { return population_density; }
Cint CCountry::getNumber_of_cities() const { return number_of_cities; }
Cint CCountry::getPopulation() const { return population; }
Cint CCountry::getArea() const { return area; }
Cint CCountry::getUnical_index() const { return unical_index; }
bool CCountry::getPlace_of_birth_citizen() const { return
citizen.getPlace_of_birth_citizen(); }
bool CCountry::getBirthday_citizen() const { return citizen.getBirthday_citizen(); }
void CCountry::setTitle(const std::string& Title) { title = Title; }
void CCountry::setPopulation_density(const int& Population_density) { population_density
= Population_density; }
void CCountry::setNumber_of_cities(const int& Number_of_cities) { number_of_cities =
Number_of_cities; }
void CCountry::setPopulation(const int& Population) { population = Population; }
void CCountry::setArea(const int& Area) { area = Area; }
void CCountry::setUnical_index(const int& Unical_index) { unical_index = Unical_index; }
void CCountry::setPlace_of_birth_citizen(const bool& Place_of_birth_citizen) {
citizen.setPlace_of_birth_citizen(Place_of_birth_citizen); }
void CCountry::setBirthday_citizen(const bool& Birthday_citizen) {
citizen.setBirthday_citizen(Birthday_citizen); }
std::string CCountry::getInfo() const
{
    return "";
}
CCountry::CCountry()
{
    title = "CCountry";
    population_density = 1000;
    number_of_cities = 100;
    population = 1000000;
    area = 10000000;
    unical_index = 0;
    citizen.setPlace_of_birth_citizen(false);
    citizen.setBirthday_citizen(false);
    std::cout << "Файл создан при помощи конструктора по умолчанию." << "\n";
}
CCountry::CCountry(const CCountry& CCountry)
{

```



```

        title = CCountry.title;
        population_density = CCountry.population_density;
        number_of_cities = CCountry.number_of_cities;
        population = CCountry.population;
        area = CCountry.area;
        unical_index = CCountry.unical_index;
        citizen = CCountry.citizen;
    }
    CCountry::CCountry(const std::string& Title, const int& Number_of_cities, const int&
    Population, const int& Area, const bool& Place_of_birth_citizen, const bool&
    Birthday_citizen, const int& Unical_index)
    {
        title = Title;
        number_of_cities = Number_of_cities;
        population = Population;
        area = Area;
        population_density = Area / Population;
        citizen.setPlace_of_birth_citizen(Place_of_birth_citizen);
        citizen.setBirthday_citizen(Birthday_citizen);
        unical_index = Unical_index;
        std::cout << "Файл создан при помощи конструктора с аргументами." << "\n";
    }
    CCountry::~~CCountry()
    {
        std::cout << "Файл уничтожен при помощи деструктора по умолчанию." << "\n";
    }
    const bool CCitizen::getPlace_of_birth_citizen() const { return place_of_birth_citizen; }
    const bool CCitizen::getBirthday_citizen() const { return birthday_citizen; }
    void CCitizen::setPlace_of_birth_citizen(const bool& Place_of_birth_citizen) {
    place_of_birth_citizen = Place_of_birth_citizen; }
    void CCitizen::setBirthday_citizen(const bool& Birthday_citizen) { birthday_citizen =
    Birthday_citizen; }
    bool Inheritor_CCountry::getMonarchy() const { return monarchy; }
    void Inheritor_CCountry::setMonarchy(const bool& Monarchy) { monarchy = Monarchy; }
    std::string Inheritor_CCountry::getInfo() const
    {
        std::stringstream s;
        s << monarchy;
        return s.str();
    }
    Inheritor_CCountry::Inheritor_CCountry() : CCountry(), monarchy(true)
    {
        type_of_Country = 1;
    }
    Inheritor_CCountry::Inheritor_CCountry(const Inheritor_CCountry& in_CC) :
    CCountry(in_CC), monarchy(in_CC.monarchy)
    {
        type_of_Country = 1;
    }
    Inheritor_CCountry::Inheritor_CCountry(const std::string& Title, const int&
    Number_of_cities, const int& Population, const int& Area, const bool&
    Place_of_birth_citizen, const bool& Birthday_citizen, const int& Unical_index, const
    bool& Monarchy) : CCountry(Title, Number_of_cities, Population, Area,
    Place_of_birth_citizen, Birthday_citizen, Unical_index), monarchy(Monarchy)
    {
        type_of_Country = 1;
    }
    Inheritor_CCountry::~~Inheritor_CCountry() { }

    bool Inheritor_CCountry_second::getGross_domestic_product() const { return
    gross_domestic_product; }
    void Inheritor_CCountry_second::setGross_domestic_product(const bool&
    Gross_domestic_product) { gross_domestic_product = Gross_domestic_product; }
    Inheritor_CCountry_second::Inheritor_CCountry_second() : CCountry(),
    gross_domestic_product(true)
    {
        type_of_Country = 2;
    }

```

```

Inheritor_CCountry_second::Inheritor_CCountry_second(const Inheritor_CCountry_second&
in_CC_second) : CCountry(in_CC_second),
gross_domestic_product(in_CC_second.gross_domestic_product)
{
    type_of_Country = 2;
}
Inheritor_CCountry_second::Inheritor_CCountry_second(const std::string& Title, const int&
Number_of_cities, const int& Population, const int& Area, const bool&
Place_of_birth_citizen, const bool& Birthday_citizen, const int& Unical_index, const
bool& Gross_domestic_product) : CCountry(Title, Number_of_cities, Population, Area,
Place_of_birth_citizen, Birthday_citizen, Unical_index),
gross_domestic_product(Gross_domestic_product)
{
    type_of_Country = 2;
}
Inheritor_CCountry_second::~Inheritor_CCountry_second() { }
std::string Inheritor_CCountry_second::getInfo() const
{
    std::stringstream s;
    s << gross_domestic_product;
    return s.str();
}
bool operator== (const CCountry& Country1, const CCountry& Country2)
{
    if (Country1.getTitle() != Country2.getTitle())
    {
        return false;
    }
    else if (Country1.getPopulation_density() != Country2.getPopulation_density())
    {
        return false;
    }
    else if (Country1.getNumber_of_cities() != Country2.getNumber_of_cities())
    {
        return false;
    }
    else if (Country1.getPopulation() != Country2.getPopulation())
    {
        return false;
    }
    else if (Country1.getArea() != Country2.getArea())
    {
        return false;
    }
    else if (Country1.getUnical_index() != Country2.getUnical_index())
    {
        return false;
    }
    else
    {
        return true;
    }
}
bool operator!= (const CCountry& Country1, const CCountry& Country2)
{
    return !(Country1 == Country2);
}
bool operator== (const Inheritor_CCountry& Inheritor_Country1, const Inheritor_CCountry&
Inheritor_Country2)
{
    if (Inheritor_Country1.getTitle() != Inheritor_Country2.getTitle())
    {
        return false;
    }
    else if (Inheritor_Country1.getPopulation_density() !=
Inheritor_Country2.getPopulation_density())
    {
        return false;
    }

```

```

    }
    else if (Inheritor_Country1.getNumber_of_cities() !=
Inheritor_Country2.getNumber_of_cities())
    {
        return false;
    }
    else if (Inheritor_Country1.getPopulation() != Inheritor_Country2.getPopulation())
    {
        return false;
    }
    else if (Inheritor_Country1.getArea() != Inheritor_Country2.getArea())
    {
        return false;
    }
    else if (Inheritor_Country1.getUnical_index() !=
Inheritor_Country2.getUnical_index())
    {
        return false;
    }
    else if (Inheritor_Country1.getMonarchy() != Inheritor_Country2.getMonarchy())
    {
        return false;
    }
    else
    {
        return true;
    }
}

bool operator!= (const Inheritor_CCountry& Inheritor_Country1, const Inheritor_CCountry&
Inheritor_Country2)
{
    return !(Inheritor_Country1 == Inheritor_Country2);
}

bool operator== (const Inheritor_CCountry_second& Inheritor_Country_second1, const
Inheritor_CCountry_second& Inheritor_Country_second2)
{
    if (Inheritor_Country_second1.getTitle() != Inheritor_Country_second2.getTitle())
    {
        return false;
    }
    else if (Inheritor_Country_second1.getPopulation_density() !=
Inheritor_Country_second2.getPopulation_density())
    {
        return false;
    }
    else if (Inheritor_Country_second1.getNumber_of_cities() !=
Inheritor_Country_second2.getNumber_of_cities())
    {
        return false;
    }
    else if (Inheritor_Country_second1.getPopulation() !=
Inheritor_Country_second2.getPopulation())
    {
        return false;
    }
    else if (Inheritor_Country_second1.getArea() !=
Inheritor_Country_second2.getArea())
    {
        return false;
    }
    else if (Inheritor_Country_second1.getUnical_index() !=
Inheritor_Country_second2.getUnical_index())
    {
        return false;
    }
    else if (Inheritor_Country_second1.getGross_domestic_product() !=
Inheritor_Country_second2.getGross_domestic_product())
    {

```

```

        return false;
    }
    else
    {
        return true;
    }
}
bool operator!=(const Inheritor_CCountry_second& Inheritor_Country_second1, const
Inheritor_CCountry_second& Inheritor_Country_second2)
{
    return !(Inheritor_Country_second1 == Inheritor_Country_second2);
}
bool check_str(const std::string& str)
{
    std::regex reg("[A-Za-zA-Яа-я0-9\\s\\!,\\?\\\"\\.\\.;\\'\\']*");
    if (!(std::regex_search(str, reg)))
    {
        return false;
    }
    std::regex reg_2("\\s{2,}");
    if (std::regex_search(str, reg_2))
    {
        return false;
    }
    std::regex reg_3("[\\!\\?\\.\\.;]{2,}");
    if (std::regex_search(str, reg_3))
    {
        return false;
    }
    std::regex reg_4("[\\'\\\"]{2,}");
    if (std::regex_search(str, reg_4))
    {
        return false;
    }
    return true;
}
std::ostream& operator<< (std::ostream& os, const CCountry& Country)
{
    return os << Country.type_of_Country << " " << "_ " << Country.getTitle() << " "
<< Country.getNumber_of_cities() << " " << Country.getPopulation() << " " <<
Country.getArea() << " " << Country.getPlace_of_birth_citizen() << " " <<
Country.getBirthday_citizen() << " " << Country.getUnical_index() << " " <<
Country.getInfo();
}
std::ostream& operator<< (std::ostream& os, const Inheritor_CCountry& Inheritor_Country)
{
    return os << Inheritor_Country.type_of_Country << " " << "_ " <<
Inheritor_Country.getTitle() << " " << Inheritor_Country.getNumber_of_cities() << " " <<
Inheritor_Country.getPopulation() << " " << Inheritor_Country.getArea() << " " <<
Inheritor_Country.getPlace_of_birth_citizen() << " " <<
Inheritor_Country.getBirthday_citizen() << " " << Inheritor_Country.getUnical_index() <<
" " << Inheritor_Country.getMonarchy();
}
std::ostream& operator<< (std::ostream& os, const Inheritor_CCountry_second&
Inheritor_Country_second)
{
    return os << Inheritor_Country_second.type_of_Country << " " << "_ " <<
Inheritor_Country_second.getTitle() << " " <<
Inheritor_Country_second.getNumber_of_cities() << " " <<
Inheritor_Country_second.getPopulation() << " " << Inheritor_Country_second.getArea() <<
" " << Inheritor_Country_second.getPlace_of_birth_citizen() << " " <<
Inheritor_Country_second.getBirthday_citizen() << " " <<
Inheritor_Country_second.getUnical_index() << " " <<
Inheritor_Country_second.getGross_domestic_product();
}
std::istream& operator>>(std::istream& is, Inheritor_CCountry& Inheritor_Country)
{
    std::string title;

```

```

std::string temp;
std::regex reg("_$");
std::stringstream temps;
Inheritor_CCountry temp_In_CC;
bool check = true;
bool global_check = true;
do
{
    is >> temp;
    if (check_str(temp))
    {
        title += temp;
    }
    else
    {
        global_check = false;
    }
    if (std::regex_search(title, reg))
    {
        check = false;
    }
    else
    {
        title += " ";
    }
} while (check);
std::regex reg_1("_");
title = std::regex_replace(title, reg_1, "");
temp_In_CC.setTitle(title);
int temp_i = 0;
is >> temp;
if (!check_str(temp))
{
    global_check = false;
}
temps << temp;
temps >> temp_i;
temps.clear();
temp_In_CC.setNumber_of_cities(temp_i);
is >> temp;
if (!check_str(temp))
{
    global_check = false;
}
temps << temp;
temps >> temp_i;
temps.clear();
temp_In_CC.setPopulation(temp_i);
is >> temp;
if (!check_str(temp))
{
    global_check = false;
}
temps << temp;
temps >> temp_i;
temps.clear();
temp_In_CC.setArea(temp_i);
is >> temp;
if (!check_str(temp))
{
    global_check = false;
}
temps << temp;
temps >> temp_i;
temps.clear();
temp_In_CC.setPlace_of_birth_citizen(temp_i);
is >> temp;
if (!check_str(temp))

```

```

{
    global_check = false;
}
temps << temp;
temps >> temp_i;
temps.clear();
temp_In_CC.setBirthday_citizen(temp_i);
is >> temp;
if (!check_str(temp))
{
    global_check = false;
}
temps << temp;
temps >> temp_i;
temps.clear();
temp_In_CC.setUnical_index(temp_i);
is >> temp;
if (!check_str(temp))
{
    global_check = false;
}
temps << temp;
temps >> temp_i;
temps.clear();
temp_In_CC.setMonarchy(temp_i);
if (global_check == true)
{
    Inheritor_Country = temp_In_CC;
}
else
{
    temp_In_CC.type_of_Country = -1;
}
return is;
}
std::istream& operator>>(std::istream& is, Inheritor_CCountry_second&
Inheritor_Country_second) {
    std::string title;
    std::string temp;
    std::regex reg("_$");
    std::stringstream temps;
    Inheritor_CCountry_second temp_In_CC_S;
    bool check = true;
    bool global_check = true;
    do {
        is >> temp;
        if (check_str(temp))
        {
            title += temp;
        }
        else {
            global_check = false;
        }
        if (std::regex_search(title, reg))
        {
            check = false;
        }
        else
        {
            title += " ";
        }
    } while (check);
    std::regex reg_1("_");
    title = std::regex_replace(title, reg_1, "");
    temp_In_CC_S.setTitle(title);
    int temp_i = 0;
    std::string temp_i_1;
    is >> temp;

```



```

if (!check_str(temp))
{
    global_check = false;
}
temps << temp;
temps >> temp_i;
temps.clear();
temp_In_CC_S.setNumber_of_cities(temp_i);
is >> temp;
if (!check_str(temp))
{
    global_check = false;
}
temps << temp;
temps >> temp_i;
temps.clear();
temp_In_CC_S.setPopulation(temp_i);
is >> temp;
if (!check_str(temp))
{
    global_check = false;
}
temps << temp;
temps >> temp_i;
temps.clear();
temp_In_CC_S.setArea(temp_i);
is >> temp;
if (!check_str(temp))
{
    global_check = false;
}
temps << temp;
temps >> temp_i;
temps.clear();
temp_In_CC_S.setPlace_of_birth_citizen(temp_i);
is >> temp;
if (!check_str(temp))
{
    global_check = false;
}
temps << temp;
temps >> temp_i;
temps.clear();
temp_In_CC_S.setBirthday_citizen(temp_i);
is >> temp;
if (!check_str(temp))
{
    global_check = false;
}
temps << temp;
temps >> temp_i;
temps.clear();
temp_In_CC_S.setUnical_index(temp_i);
is >> temp;
if (!check_str(temp))
{
    global_check = false;
}
temps << temp;
temps >> temp_i;
temps.clear();
temp_In_CC_S.setGross_domestic_product(temp_i);
if (global_check == true)
{
    Inheritor_Country_second = temp_In_CC_S;
}
else
{

```

```

        Inheritor_Country_second.type_of_Country = -1;
    }
    return is;
}
CCountry& CCountry::operator= (const CCountry& Country)
{
    title = Country.title;
    population_density = Country.population_density;
    number_of_cities = Country.number_of_cities;
    population = Country.population;
    area = Country.area;
    unical_index = Country.unical_index;
    citizen.setPlace_of_birth_citizen(Country.getPlace_of_birth_citizen());
    citizen.setBirthday_citizen(Country.getBirthday_citizen());
    return *this;
}
Inheritor_CCountry& Inheritor_CCountry::operator= (const Inheritor_CCountry&
Inheritor_Country)
{
    title = Inheritor_Country.title;
    population_density = Inheritor_Country.population_density;
    number_of_cities = Inheritor_Country.number_of_cities;
    population = Inheritor_Country.population;
    area = Inheritor_Country.area;
    unical_index = Inheritor_Country.unical_index;
    citizen.setPlace_of_birth_citizen(Inheritor_Country.getPlace_of_birth_citizen());
    citizen.setBirthday_citizen(Inheritor_Country.getBirthday_citizen());
    monarchy = Inheritor_Country.monarchy;
    return *this;
}
Inheritor_CCountry_second& Inheritor_CCountry_second::operator=(const
Inheritor_CCountry_second& Inheritor_Country_second)
{
    title = Inheritor_Country_second.title;
    population_density = Inheritor_Country_second.population_density;
    number_of_cities = Inheritor_Country_second.number_of_cities;
    population = Inheritor_Country_second.population;
    area = Inheritor_Country_second.area;
    unical_index = Inheritor_Country_second.unical_index;
    citizen.setPlace_of_birth_citizen(Inheritor_Country_second.getPlace_of_birth_citizen());
    citizen.setBirthday_citizen(Inheritor_Country_second.getBirthday_citizen());
    gross_domestic_product = Inheritor_Country_second.gross_domestic_product;
    return *this;
}
bool operator> (const CCountry& Country1, const CCountry& Country2) {
    return Country1.getTitle() < Country2.getTitle();
}
bool operator< (const CCountry& Country1, const CCountry& Country2) {
    return Country1.getTitle() > Country2.getTitle();
}
bool operator> (const Inheritor_CCountry& Inheritor_Country1, const Inheritor_CCountry&
Inheritor_Country2) {
    return Inheritor_Country1.getTitle() < Inheritor_Country2.getTitle();
}
bool operator< (const Inheritor_CCountry& Inheritor_Country1, const Inheritor_CCountry&
Inheritor_Country2) {
    return Inheritor_Country1.getTitle() > Inheritor_Country2.getTitle();
}
bool operator> (const Inheritor_CCountry_second& Inheritor_Country_second1, const
Inheritor_CCountry_second& Inheritor_Country_second2) {
    return Inheritor_Country_second1.getTitle() <
Inheritor_Country_second2.getTitle();
}
bool operator< (const Inheritor_CCountry_second& Inheritor_Country_second1, const
Inheritor_CCountry_second& Inheritor_Country_second2) {

```

```

        return Inheritor_Country_second1.getTitle() >
Inheritor_Country_second2.getTitle();
}
CMethod.h
#pragma once
#include "CCountry.h"
class CMethod
{
private:
    CCountry** countries;
    CCountry** copy;
    CInt next_i = 0;
    CInt new_i = 1;
public:
    void add_el(const Inheritor_CCountry& Inheritor_Country);
    void add_el(const Inheritor_CCountry_second& Inheritor_Country_second);
    void remove_el(const int& index);
    void del_all();
    CCountry* find_to_index(const int& index) const;
    void print_all() const;
    void find_to_population_density() const;
    void find_to_str_by_file(const std::string str);
    std::string get_str_by_file(const int& index) const;
    void write_to_file(const std::string name);
    void read_from_file(const std::string name);
    bool check_str(const std::string& str) const;
    void print_all_with_2_or_more_words() const;
    void sort(bool (*comp)(CCountry&, CCountry&));
    CCountry* operator[](const int& index)
    {
        return countries[index];
    }
    friend std::ostream& operator<< (std::ostream& os, CMethod& CMethod);
};

struct count_place_of_birth_citizen
{
    count_place_of_birth_citizen() { count = 0; }
    void operator() (CCountry Country);
    int count;
};

struct count_birthday_citizen
{
    count_birthday_citizen() { count = 0; }
    void operator() (CCountry Country);
    int count;
};

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

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

bool sortTitle(CCountry&, CCountry&);
bool sortNumber_of_cities(CCountry&, CCountry&);
bool sortPopulation(CCountry&, CCountry&);
bool sortArea(CCountry&, CCountry&);
bool sortCitizen(CCountry&, CCountry&);
std::istream& operator>> (std::istream& is, CMethod& CMethod);
std::ostream& operator<< (std::ostream& os, CMethod& CMethod);
void print(const CCountry& Country);

```

```

void pprint(const std::pair<int, CCountry>& p);
void print_place_of_birth_citizen(const CCountry& Country);
void print_birthday_citizen(const CCountry& Country);
void pprint_place_of_birth_citizen(const std::pair<int, CCountry>& p);
void pprint_birthday_citizen(const std::pair<int, CCountry>& p);
CMetod.cpp
#include "CMetod.h"

void CMetod::add_el(const Inheritor_CCountry& Inheritor_Country)
{
    if (next_i == 0)
    {
        countries = new CCountry * [next_i + 1];
        CCountry* point1 = new auto(Inheritor_Country);
        countries[next_i] = point1;
        next_i++;
        new_i++;
    }
    else
    {
        copy = new CCountry * [next_i + 1];
        for (int i = 0; i < next_i; i++)
        {
            copy[i] = countries[i];
        }
        delete[] countries;
        countries = new CCountry * [next_i + 1];
        for (int i = 0; i < next_i; i++)
        {
            countries[i] = copy[i];
        }
        CCountry* point1 = new auto(Inheritor_Country);
        countries[next_i] = point1;
        delete[] copy;
        next_i++;
        new_i++;
    }
}

void CMetod::add_el(const Inheritor_CCountry_second& Inheritor_Country_second)
{
    if (next_i == 0)
    {
        countries = new CCountry * [next_i + 1];
        CCountry* point2 = new auto(Inheritor_Country_second);
        countries[next_i] = point2;
        next_i++;
        new_i++;
    }
    else
    {
        copy = new CCountry * [next_i + 1];
        for (int i = 0; i < next_i; i++)
        {
            copy[i] = countries[i];
        }
        delete[] countries;
        countries = new CCountry * [next_i + 1];
        for (int i = 0; i < next_i; i++)
        {
            countries[i] = copy[i];
        }
        CCountry* point2 = new auto(Inheritor_Country_second);
        countries[next_i] = point2;
        delete[] copy;
        next_i++;
        new_i++;
    }
}

```

```

void CMetod::remove_el(const int& index)
{
    if (next_i == 1)
    {
        delete[] countries;
        next_i--;
    }
    else
    {
        copy = new CCountry * [next_i - 1];
        for (int i = 0; i < index; i++)
        {
            copy[i] = countries[i];
        }
        for (int i = index, j = index + 1; i < (next_i - 1), j < next_i; i++, j++)
        {
            copy[i] = countries[j];
        }
        delete[] countries;
        countries = new CCountry * [next_i - 1];
        for (int i = 0; i < next_i - 1; i++)
        {
            countries[i] = copy[i];
        }
        delete[] copy;
        next_i--;
    }
}

void CMetod::del_all()
{
    if (next_i != 0)
    {
        for (int i = 0; i < next_i; i++)
        {
            delete countries[i];
        }
        delete[] countries;
        next_i = 0;
    }
}

CCountry* CMetod::find_to_index(const int& index) const
{
    for (int i = 0; i < next_i; i++)
    {
        if (countries[i]->getUnical_index() == index)
        {
            return countries[i];
        }
    }
}

void CMetod::print_all() const
{
    for (int i = 0; i < next_i; i++)
    {
        std::cout << i + 1 << " " << (*(countries + i)) << "\n";
    }
}

void CMetod::find_to_population_density() const
{
    float min = countries[0]->getPopulation_density();
    for (int i = 0; i < next_i; i++)
    {
        if (min > countries[i]->getPopulation_density())
        {
            min = countries[i]->getPopulation_density();
        }
    }
}

```

```

    for (int i = 0; i < next_i; i++)
    {
        if (countries[i]->getPopulation_density() == min)
            std::cout << get_str_by_file(i) << "\n";
    }
}
std::string CMethod::get_str_by_file(const int& index) const
{
    std::stringstream ss;
    ss << countries[index];
    return ss.str();
}
void CMethod::find_to_str_by_file(const std::string str)
{
    if (check_str(str))
    {
        std::regex reg("_.+");
        std::smatch smat;
        std::regex_search(str, smat, reg);
        int i = str.find("_");
        i = str.find(" ", i + 1);
        std::regex reg_temp("_");
        std::string temp = smat[0];
        std::string Title = std::regex_replace(temp, reg_temp, "_");
        int i2 = str.find(" ", i + 2);
        temp = str.substr(i + 1, i2 - i);
        std::stringstream s;
        s << temp;
        int Number_of_cities;
        s >> Number_of_cities;
        int i3 = str.find(" ", i2 + 1);
        s.clear();
        temp = str.substr(i2 + 1, i3 - i2);
        s << temp;
        int Population;
        s >> Population;
        int i4 = str.find(" ", i3 + 1);
        s.clear();
        temp = str.substr(i3 + 1, i4 - i3);
        s << temp;
        int Area;
        s >> Area;
        int i5 = str.find(" ", i4 + 1);
        s.clear();
        temp = str.substr(i4 + 1, i5 - i4);
        s << temp;
        bool Place_of_birth_citizen;
        s >> Place_of_birth_citizen;
        int i6 = str.find(" ", i5 + 1);
        s.clear();
        temp = str.substr(i5 + 1, i6 - i5);
        s << temp;
        bool Birthday_citizen;
        s >> Birthday_citizen;
        int i7 = str.find(" ", i6 + 1);
        s.clear();
        temp = str.substr(i6 + 1, i7 - i6);
        s << temp;
        int Unical_index;
        s >> Unical_index;
        int i8 = str.find(" ", i7 + 1);
        s.clear();
        temp = str.substr(i7 + 1, i8 - i7);
        s << temp;
        s.clear();
        int check;
        s >> check;
        if (check == 1)
    }
}

```



```

        {
            bool Monarchy;
            s >> Monarchy;
            Inheritor_CCountry firstcountry(Title, Number_of_cities, Population,
Area, Place_of_birth_citizen, Birthday_citizen, Unical_index, Monarchy);
            add_el(firstcountry);
        }
        else
        {
            bool Gross_domestic_product;
            s >> Gross_domestic_product;
            Inheritor_CCountry_second secondcountry(Title, Number_of_cities,
Population, Area, Place_of_birth_citizen, Birthday_citizen, Unical_index,
Gross_domestic_product);
            add_el(secondcountry);
        }
    }
}

void CMetod::write_to_file(const std::string name)
{
    std::ofstream fout("text.txt");
    std::string s;
    for (int i = 0; i < next_i; i++)
    {
        s = get_str_by_file(i);
        fout << s;
        if (i != next_i - 1)
        {
            fout << "\n";
        }
    }
    fout.close();
}

void CMetod::read_from_file(const std::string name)
{
    del_all();
    std::ifstream fin("text.txt");
    char* check;
    while (!fin.eof())
    {
        check = new char[100];
        fin.getline(check, 100);
        find_to_str_by_file(check);
        delete[] check;
    }
    fin.close();
}

bool CMetod::check_str(const std::string& str) const
{
    std::regex reg("[A-Za-zA-Яa-я0-9\\!,\\?\\\"\\.\\:;\\'\\\"]*");
    if (!(std::regex_search(str, reg)))
    {
        return false;
    }
    std::regex reg_2("\\s{2,}");
    if (std::regex_search(str, reg_2))
    {
        return false;
    }
    std::regex reg_3("[\\!\\?\\:\\.\\,\\;]{2,}");
    if (std::regex_search(str, reg_3))
    {
        return false;
    }
    std::regex reg_4("[\\'\\\"]{2,}");
    if (std::regex_search(str, reg_4))
    {
        return false;
    }
}

```

```

    }
    std::regex reg_5("^\"[A-ZА-Я]");
    if (!std::regex_search(str, reg_5))
    {
        return false;
    }
    return true;
}
void CMethod::print_all_with_2_or_more_words() const
{
    for (int i = 0; i < next_i; i++)
    {
        std::regex reg("_.+ .+_");
        std::string str = countries[i]->getTitle();
        if (std::regex_search(str, reg))
        {
            std::cout << i + 1 << " " << (*(countries + i)) << "\n";
        }
    }
}
void CMethod::sort(bool (*comp)(CCountry&, CCountry&))
{
    bool pr = false;
    CCountry* temp;
    do
    {
        pr = false;
        for (int i = 0; i < next_i - 1; i++)
        {
            if (comp(*(countries[i]), *(countries[i + 1])))
            {
                temp = countries[i];
                countries[i] = countries[i + 1];
                countries[i + 1] = temp;
                pr = true;
            }
        }
    } while (pr);
}
bool sortTitle(CCountry& a, CCountry& b)
{
    return (a.getTitle() > b.getTitle());
}
bool sortNumber_of_cities(CCountry& a, CCountry& b)
{
    return(a.getNumber_of_cities() < b.getNumber_of_cities());
}
bool sortPopulation(CCountry& a, CCountry& b)
{
    return(a.getPopulation() < b.getPopulation());
}
bool sortArea(CCountry& a, CCountry& b)
{
    return (a.getArea() < b.getArea());
}
bool sortCitizen(CCountry& a, CCountry& b)
{
    return((a.getPlace_of_birth_citizen() > b.getPlace_of_birth_citizen()) &&
(a.getBirthday_citizen() < b.getBirthday_citizen()));
}
std::istream& operator>> (std::istream& is, CMethod& Cmethod) {
    int temp;
    Inheritor_CCountry In_CC;
    Inheritor_CCountry_second In_CC_S;
    while (is >> temp) {
        if (temp == 1) {
            is >> In_CC;
            if (In_CC.type_of_Country != -1) {

```

```

        Cmetod.add_el(In_CC);
    }
    }
    else {
        is >> In_CC_S;
        if (In_CC_S.type_of_Country != -1) {
            Cmetod.add_el(In_CC_S);
        }
    }
}
return is;
}
std::ostream& operator<<(std::ostream& os, CMetod& Cmetod)
{
    for (size_t i = 0; i < Cmetod.next_i; i++)
    {
        os << *(Cmetod[i]) << "\n";
    }
    return os;
}
void count_place_of_birth_citizen::operator()(CCountry Country)
{
    if (Country.getPlace_of_birth_citizen() == true)
    {
        count++;
    }
}
void count_birthday_citizen::operator()(CCountry Country)
{
    if (Country.getBirthday_citizen() == 64)
    {
        count++;
    }
}
void print(const CCountry& Country)
{
    std::cout << Country << "\n";
}
void pprint(const std::pair<int, CCountry>& p)
{
    std::cout << p.first << " : " << p.second << "\n";
}
void print_place_of_birth_citizen(const CCountry& Country)
{
    if (Country.getPlace_of_birth_citizen() == true)
    {
        std::cout << Country << "\n";
    }
}
void print_birthday_citizen(const CCountry& Country)
{
    if (Country.getBirthday_citizen() == true)
    {
        std::cout << Country << "\n";
    }
}
void ccount_place_of_birth_citizen::operator()(std::pair<int, CCountry> p)
{
    if (p.second.getPlace_of_birth_citizen() == true)
    {
        count++;
    }
}
void ccount_birthday_citizen::operator()(std::pair<int, CCountry> p)
{
    if (p.second.getBirthday_citizen() == true)
    {
        count++;
    }
}

```

```

    }
}
void pprint_place_of_birth_citizen(const std::pair<int, CCountry>& p)
{
    if (p.second.getPlace_of_birth_citizen() == true)
    {
        std::cout << p.first << " : " << p.second << "\n";
    }
}
void pprint_birthday_citizen(const std::pair<int, CCountry>& p)
{
    if (p.second.getBirthday_citizen() == true)
    {
        std::cout << p.first << " : " << p.second << "\n";
    }
}

```

```

menu.h
#pragma once
#include "CMetod.h"

void old_menu();
void menu();
menu.cpp
#include "menu.h"
void old_menu()
{
    setlocale(LC_ALL, "Russian"); /// Локализация консоли.
    int n = 0, temp_i;
    CMetod dir;
    std::ifstream f("text.txt");
    std::ofstream d;
    f >> dir;
    f.close();
    int c;
    while (n != 9)
    {
        std::cout << "-----МЕНЮ-----" << "\n";
        std::cout << "-----" << "\n";
        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 - найти наименьшую плотность населения страны." << "\n";
        std::cout << "-----5 - записать данные а файл.-----" << "\n";
        std::cout << "-----6 - считать данные из файла.-----" << "\n";
        std::cout << "-----7 - найти все элеметы в названии которых есть 2 или больше
слова." << "\n";
        std::cout << "-----8 - Отсортировать массив.-----" << "\n";
        std::cout << "-----9 - завершить работу программы.-----" << "\n";
        std::cout << "-----" << "\n";
        std::cin >> n;
        if (n == 1)
        {
            dir.find_to_str_by_file("_ Страна5_ 323 93645665 78767464 1 24112001 0 1");
            std::cout << "Страна добавлена." << "\n";
        }
        else if (n == 2)
        {
            std::cout << "Введите номер удаляемого элемента (нумерация начинается с 1):
";
            std::cin >> temp_i;
            dir.remove_el(temp_i - 1);
            std::cout << "Страна удалена." << "\n";
        }
    }
}

```

```

else if (n == 3)
{
    std::cout << dir;
}
else if (n == 4)
{
    dir.find_to_population_density();
}
else if (n == 5)
{
    d.open("text.txt");
    d << dir;
    d.close();
}
else if (n == 6)
{
    f.open("text.txt");
    f >> dir;
    f.close();
}
else if (n == 7)
{
    dir.print_all_with_2_or_more_words();
}
else if (n == 8)
{
    std::cout << "Введите номер признака, по которому хотите отсортировать
массив: 1 - title, 2 - number_of_cities, 3 - population, 4 - area, 5 - citizen." << "\n";
    std::cin >> c;
    if (c == 1)
    {
        dir.sort(sortTitle);
    }
    else if (c == 2)
    {
        dir.sort(sortNumber_of_cities);
    }
    else if (c == 3)
    {
        dir.sort(sortPopulation);
    }
    else if (c == 4)
    {
        dir.sort(sortArea);
    }
    else if (c == 5)
    {
        dir.sort(sortCitizen);
    }
    else
    {
        std::cout << "Неправильный номер." << "\n";
        n = 0;
        break;
    }
}
}
dir.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<CCountry> vector;
    Inheritor_CCountry Intemp;
    Inheritor_CCountry_second Instemp;
    std::ifstream fin("text.txt");
    while (fin >> n)
    {
        if (n == 1)
        {
            fin >> Intemp;
            vector.push_back(Intemp);
            vector[q].setUnical_index(q);
        }
        else
        {
            fin >> Instemp;
            vector.push_back(Instemp);
            vector[q].setUnical_index(q);
        }
        q++;
    }
    fin.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 - завершить работу программы.\n";
        std::cout << "Введите число, что соответствует необходимому действию: ";
        std::cin >> n;
        if (n == 1)
        {
            std::cout << "Вот данные вектора: \n";
            for (int i = 0; i < q; i++)
            {
                std::cout << vector[i] << "\n";
            }
        }
        else if (n == 2)
        {
            vector.clear();
            fin.open("text.txt");
            q = 0;
            while (fin >> w)
            {
                if (n == 1)
                {
                    fin >> Intemp;
                    vector.push_back(Intemp);
                    vector[q].setUnical_index(q);
                }
                else
                {
                    fin >> Instemp;
                    vector.push_back(Instemp);
                    vector[q].setUnical_index(q);
                }
            }
        }
    }
}

```



```

    }
    q++;
}
fin.close();
std::cout << "Данные считаны из файла в вектор.\n";
}
else if (n == 3)
{
    Inheritor_CCountry_second Ins("Страна0", 123, 45645645, 45756757, 6, 1,
28122009, 1);
    vector.push_back(Ins);
    vector[q].setUnical_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++)
        {
            vector[i] = vector[i + 1];
            vector[i].setUnical_index(i);
        }
        vector.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 << vector[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 - место рождения жителя города.\n";
    std::cout << "2 - дата рождения жителя города.\n";
    std::cout << "Введите значение соответствующее нужному параметру: ";
    std::cin >> n;
    if (n == 1) {
        count_place_of_birth_citizen c = std::for_each(vector.begin(),
vector.end(), count_place_of_birth_citizen());
        std::cout << "Найдено " << c.count << " элементов с заданным
параметром.\n";
    }
    else if (n == 2) {
        count_birthday_citizen c = std::for_each(vector.begin(),
vector.end(), count_birthday_citizen());
        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 - место рождения жителя города.\n";
    std::cout << "2 - дата рождения жителя города.\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(vector.begin(), vector.end(),
print_place_of_birth_citizen);
    }
    else if (n == 2)
    {
        std::cout << "Вот данные о элементах с заданной характеристикой:\n";
        std::for_each(vector.begin(), vector.end(), print_birthday_citizen);
    }
    else
    {
        std::cout << "Такого параметра нет.\n";
    }
}
else
{
    break;
}
}
}
else if (n == 2)
{
    bool check;
    std::string title;
    std::list<CCountry> list;
    Inheritor_CCountry Intemp;
    Inheritor_CCountry_second Instemp;
    std::ifstream fin("text.txt");
    while (fin >> n)
    {
        if (n == 1)
        {
            fin >> Intemp;
            list.push_back(Intemp);
        }
        else
        {
            fin >> Instemp;
            list.push_back(Instemp);
        }
    }
    fin.close();
    std::cout << "Данные считаны из файла в список.\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 - 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::copy(list.begin(), list.end(),
std::ostream_iterator<CCountry>(std::cout, " \n"));
}
else if (n == 2)
{
    list.clear();
    fin.open("text.txt");
    while (fin >> n)
    {
        if (n == 1)
        {
            fin >> Intemp;
            list.push_back(Intemp);
        }
        else
        {
            fin >> Instemp;
            list.push_back(Instemp);
        }
    }
    fin.close();
    std::cout << "Данные считаны из файла в список.\n";
}
else if (n == 3)
{
    Inheritor_CCountry_second Ins("Страна0", 123, 45645645, 45756757, 6, 1,
28122009, 1);
    list.push_back(Ins);
    std::cout << "Элемент добавлен в список.\n";
}
else if (n == 4)
{
    std::cout << "Введите название удаляемого элемента: ";
    std::cin.ignore();
    std::getline(std::cin, title);
    check = false;
    for (auto element : list)
    {
        if (element.getTitle() == title)
        {
            list.remove(element);
            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, title);
    check = true;
    for (auto element : list)

```

```

    {
        if (element.getTitle() == title)
        {
            std::cout << "Вот нужный элемент: ";
            std::cout << element << "\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 - место рождения жителя города.\n";
    std::cout << "2 - дата рождения жителя города.\n";
    std::cout << "Введите значение соответствующее нужному параметру: ";
    std::cin >> n;
    if (n == 1) {
        count_place_of_birth_citizen c = std::for_each(list.begin(),
list.end(), count_place_of_birth_citizen());
        std::cout << "Найдено " << c.count << " элементов с заданным
параметром.\n";
    }
    else if (n == 2) {
        count_birthday_citizen c = std::for_each(list.begin(), list.end(),
count_birthday_citizen());
        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 - место рождения жителя города.\n";
    std::cout << "2 - дата рождения жителя города.\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(list.begin(), list.end(),
print_place_of_birth_citizen);
    }
    else if (n == 2)
    {
        std::cout << "Вот данные о элементах с заданной характеристикой:\n";
        std::for_each(list.begin(), list.end(), print_birthday_citizen);
    }
    else
    {
        std::cout << "Такого параметра нет.\n";
    }
}
else
{
    break;
}
}
}

```

```

}
else if (n == 3)
{
    int q = 0;
    bool check;
    std::map<int, CCountry> mp;
    Inheritor_CCountry Intemp;
    Inheritor_CCountry_second Instemp;
    std::ifstream file("text.txt");
    while (file >> n)
    {
        if (n == 1)
        {
            file >> Intemp;
            mp.insert(std::pair<int, CCountry>(q++, Intemp));
        }
        else
        {
            file >> Instemp;
            mp.insert(std::pair<int, CCountry>(q++, Instemp));
        }
    }
    file.close();
    std::cout << "Данные считаны из файла в контейнер.\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 - 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";
            for (auto element : mp)
            {
                std::cout << element.first << " : " << element.second << "\n";
            }
        }
        else if (n == 2)
        {
            mp.clear();
            file.open("text.txt");
            while (file >> n)
            {
                if (n == 1)
                {
                    file >> Intemp;
                    mp.insert(std::pair<int, CCountry>(q++, Intemp));
                }
                else
                {
                    file >> Instemp;
                    mp.insert(std::pair<int, CCountry>(q++, Instemp));
                }
            }
            file.close();
            std::cout << "Данные считаны из файла в контейнер.\n";
        }
        else if (n == 3)
        {

```

```

28122009, 1);
    Inheritor_CCountry_second non("Страна0", 123, 45645645, 45756757, 6, 1,
mp.insert(std::pair<int, CCountry>(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 element : mp)
    {
        if (element.first == w)
        {
            mp.erase(w);
            check = true;
            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 element : mp)
    {
        if (element.first == w)
        {
            std::cout << "Вот данные нужного элемента: " << element.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 - место рождения жителя города.\n";
    std::cout << "2 - дата рождения жителя города.\n";
    std::cout << "Введите значение соответствующее нужному параметру: ";
    std::cin >> n;
    if (n == 1)
    {
        ccount_place_of_birth_citizen c = std::for_each(mp.begin(), mp.end(),
ccount_place_of_birth_citizen());
        std::cout << "Найдено " << c.count << " элементов с заданным
параметром.\n";
    }
    else if (n == 2)
    {
        ccount_birthday_citizen c = std::for_each(mp.begin(), mp.end(),
ccount_birthday_citizen());
        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 - место рождения жителя города.\n";
    std::cout << "2 - дата рождения жителя города.\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_place_of_birth_citizen);
    }
    else if (n == 2)
    {
        std::cout << "Вот данные о элементах с заданной характеристикой:\n";
        std::for_each(mp.begin(), mp.end(), pprint_birthdate_citizen);
    }
    else
    {
        std::cout << "Такого параметра нет.\n";
    }
}
else
{
    break;
}
}
}
else if (n == 4)
{
    bool check;
    std::string title;
    std::set<CCountry> st;
    Inheritor_CCountry Intemp;
    Inheritor_CCountry_second Instemp;
    std::ifstream file("text.txt");
    while (file >> n)
    {
        if (n == 1)
        {
            file >> Intemp;
            st.insert(Intemp);
        }
        else
        {
            file >> Instemp;
            st.insert(Instemp);
        }
    }
    file.close();
    std::cout << "Данные считаны из файла в множество.\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 - 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";
    for (auto element : st)
    {
        std::cout << element << "\n";
    }
}
else if (n == 2)
{
    st.clear();
    file.open("text.txt");
    while (file >> n)
    {
        if (n == 1)
        {
            file >> Intemp;
            st.insert(Intemp);
        }
        else
        {
            file >> Instemp;
            st.insert(Instemp);
        }
    }
    file.close();
    std::cout << "Данные считаны из файла в множество.\n";
}
else if (n == 3)
{
    Inheritor_CCountry_second non("Страна0", 123, 45645645, 45756757, 6, 1,
28122009, 1);
    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, title);
    check = false;
    for (auto element : st)
    {
        if (element.getTitle() == title)
        {
            st.erase(element);
            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, title);
    check = true;
}

```

```

for (auto element : st)
{
    if (element.getTitle() == title)
    {
        std::cout << "Вот ваш элемент: ";
        std::cout << element << "\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 - место рождения жителя города.\n";
    std::cout << "2 - дата рождения жителя города.\n";
    std::cout << "Введите значение соответствующее нужному параметру: ";
    std::cin >> n;
    if (n == 1)
    {
        count_place_of_birth_citizen c = std::for_each(st.begin(), st.end(),
count_place_of_birth_citizen());
        std::cout << "Найдено " << c.count << " элементов с заданным
параметром.\n";
    }
    else if (n == 2)
    {
        count_birthday_citizen c = std::for_each(st.begin(), st.end(),
count_birthday_citizen());
        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 - место рождения жителя города.\n";
    std::cout << "2 - дата рождения жителя города.\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_place_of_birth_citizen);
    }
    else if (n == 2)
    {
        std::cout << "Вот данные о элементах с заданной характеристикой:\n";
        std::for_each(st.begin(), st.end(), print_birthday_citizen);
    }
    else
    {
        std::cout << "Такого параметра нет.\n";
    }
}
}
else
{
    break;
}
}

```

```

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

int main()
{
    setlocale(LC_ALL, "Russian");
    Inheritor_CCountry In("Тест", 123, 45645645, 45756757, 6, 1, 28122009, 1);
    Inheritor_CCountry_second Ins("Тест1", 123, 45645645, 45756757, 6, 1, 28122009,
1);
    count_place_of_birth_citizen c1;
    count_birthday_citizen c2;
    c1(In);
    c2(In);
    c1(Ins);
    c2(Ins);
    std::cout << "Тест на работу функторов будет пройден если на следующей строке
будет выведен такой текст: 2 0\n" << c1.count << " " << c2.count << "\n";
    if (_CrtDumpMemoryLeaks())
    {
        std::cout << "Утечка памяти обнаружена." << "\n";
    }
    else
    {
        std::cout << "Утечка памяти не обнаружена." << "\n";
    }
    int n;
    std::cin >> n;
}
text.txt
1 _Страна1_ 143 45745656 47342362 1 1 2 0
2 _Страна2_ 156 38567454 68457458 0 0 3 1
1 _Страна3_ 167 46357625 98686453 1 1 4 0
2 _Страна4_ 179 78567583 68457458 0 0 5 1

```

4. Результати роботи програми

Результати роботи програми:

```

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 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 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 - 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.
Файл уничтожен при помощи деструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Тест на работу функторов будет пройден если на следующей строке будет выведен такой текст: 2 0
2 0
Утечка памяти не обнаружена.

```

5. Висновки

При виконанні даної лабораторної роботи було створено функтори для роботи з об'єктами класа `file` через цикл `std::for_each`, який було використано для проходження усіх елементів контейнерів.

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