Звіт

Автор: Капелька Я.І. КІТ-119а Дата: 19 травня 2020

Лабораторна робота №5. Агрегація та композиція

Тема. Класи. Агрегація. Композиція. Ключові слова *typedef* та *auto*.

Мета: отримати поняття агрегація та композиція; отримати знання про

призначення ключових слів typedef та auto.

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

Змінити попередню лабораторну роботу й продемонструвати агрегацію і композицію.

2. Опис класів, змінних, методів та функцій

2.1 Опис класів

Базовий клас: CCountry

Клас, що має в собі масив базового класу та методи для роботи з ним: CMetod

Клас, що повинен демонструвати агрегацію: CNationality

Клас, що повинен демонструвати композицію: CCitizen

2.2 Опис змінних

`Cint kol_nationality` – поле класу CNationality (кількість національностей).
`std::string place_of_birth_citizen` – поле класу CCitizen(місце народження жителя міста).
`Cint birthday_citizen` – поле класу CCitizen(дата народження жителя міста).
`CNationality* kol_nationality` – поле класу CCountry(кількість національностей).
`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).

2.3 Опис методів

`Cint getKol_Nationality() const` – отримання значення поля kol_nationality змінної класу CNationality(метод класу CNationality).

`void setKol_Nationality(CNationality* New_Kol_Nationality)` – зміна значення поля kol_nationality змінної класу CNationality (метод класу CNationality).

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

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

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

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

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

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

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

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

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

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

`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).

2.4 Опис функцій

`void menu()` – функція меню.

`bool sortTitle(CCountry&, CCountry&)` – функція порівняння двух країн по їх назві.
`bool sortNumber_of_cities(CCountry&, CCountry&)` – функція порівняння двух країн по їх кількість міст.
`bool sortPopulation(CCountry&, CCountry&)` – функція порівняння двух країн по їх популяції.
`bool sortArea(CCountry&, CCountry&)` – функція порівняння двух країн по їх площі.
`bool sortCitizen(CCountry&, CCountry&)` – функція порівняння двух країн по їх місцю та дню народження жителя міста.
`bool sortTitle_2(CCountry&, CCountry&)` – обернена функція порівняння двух країн по їх назві.
`bool sortNumber_of_cities_2(CCountry&, CCountry&)` – обернена функція порівняння двух країн по їх кількості міст.
`bool sortPopulation_2(CCountry&, CCountry&)` – обернена функція порівняння двух країн по їх популяції.
`bool sortArea_2(CCountry&, CCountry&)` – обернена функція порівняння двух країн по їх площі.
`bool sortCitizen_2(CCountry&, CCountry&)`– обернена функція порівняння двух країн по їх місцю та дню народження жителя міста.

3 Текст програми

Лабораторная работа №5.cpp

```
##pragma once
// Лабораторная работа №5.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.
//

#include <iostream>
#include "menu.h"
#define _CRTDBG_MAP_ALLOC

int main()
{
    menu();
    if (_CrtDumpMemoryLeaks())
    {
        std::cout << "Утечка памяти обнаружена." << "\n";
    }
    else
    {
        std::cout << "Утечка памяти не обнаружена." << "\n";
    }
}
```

CCountry.h

```
#pragma once
#include <iostream>
#include <string>
#include <fstream>
#include <sstream>
#include <regex>
#include <cstddef>
typedef int Cint; //Cint (Country)
class CNationality //Aggregation
{
private:
    Cint kol_nationality;
public:
    Cint getKol_Nationality() const;
    void setKol_Nationality(const int&);
};
class CCitizen //Composition
```

```
{
private:
    std::string place_of_birth_citizen;
    Cint birthday_citizen;
public:
    const std::string getPlace_of_birth_citizen() const;
    Cint getBirthday_citizen() const;
    void setPlace_of_birth_citizen(const std::string&);
    void setBirthday_citizen(const int&);
};
class CCountry
{
private:
    std::string title;
    Cint population_density;
    Cint number_of_cities;
    Cint population;
    Cint area;
    Cint unical_index;
    CNationality* kol_nationality;
    CCitizen citizen;
public:
    CCountry();
    CCountry(const CCountry&);
    CCountry(const std::string&, const int&, const int&, const int&, const std::string&, const
int&);
    ~CCountry();
    Cint getKol_Nationality() const;
    const std::string getPlace_of_birth_citizen() const;
    Cint getBirthday_citizen() const;
    const std::string getTitle() const;
    Cint getPopulation_density() const;
    Cint getNumber_of_cities() const;
    Cint getPopulation() const;
    Cint getArea() const;
    Cint getUnical_index() const;
    void setTitle(const std::string&);
    void setPopulation_density(const int&);
    void setNumber_of_cities(const int&);
    void setPopulation(const int&);
    void setArea(const int&);
    void setUnical_index(const int&);
    void setKol_Nationality(CNationality*);
    void setPlace_of_birth_citizen(const std::string&);
    void setBirthday_citizen(const int&);
};
```

CCountry.cpp

```
#include "CCountry.h"
const 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; }
Cint CCountry::getKol_Nationality() const { return kol_nationality->getKol_Nationality(); }
const std::string CCountry::getPlace_of_birth_citizen() const { return
citizen.getPlace_of_birth_citizen(); }
Cint 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::setKol_Nationality(CNationality* New_Kol_Nationality) { kol_nationality =
New_Kol_Nationality; }
```

```
void CCountry::setPlace_of_birth_citizen(const std::string& Place_of_birth_citizen) {
citizen.setPlace_of_birth_citizen(Place_of_birth_citizen); }
void CCountry::setBirthday_citizen(const int& Birthday_citizen) {
citizen.setBirthday_citizen(Birthday_citizen); }
CCountry::CCountry()
{
	title = "CCountry";
	population_density = 1000;
	number_of_cities = 100;
	population = 1000000;
	area = 10000000;
	unical_index = 0;
	citizen.setPlace_of_birth_citizen("New_York");
	citizen.setBirthday_citizen(11111111);
	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 std::string& Place_of_birth_citizen, const int& Birthday_citizen)
{
	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);
	std::cout << "Файл создан при помощи конструктора с аргументами." << "\n";
}
CCountry::~CCountry()
{
	std::cout << "Файл уничтожен при помощи деструктора по умолчанию." << "\n";
}
Cint CNationality::getKol_Nationality() const { return kol_nationality; }
void CNationality::setKol_Nationality(const int& Kol_Nationality) { kol_nationality =
Kol_Nationality; }
const std::string CCitizen::getPlace_of_birth_citizen() const { return place_of_birth_citizen; }
Cint CCitizen::getBirthday_citizen() const { return birthday_citizen; }
void CCitizen::setPlace_of_birth_citizen(const std::string& Place_of_birth_citizen) {
place_of_birth_citizen = Place_of_birth_citizen; }
void CCitizen::setBirthday_citizen(const int& Birthday_citizen) { birthday_citizen =
Birthday_citizen; }
```

CMetod.h

```
#include "CCountry.h"
class CMetod
{
private:

    CCountry* countries;
    CCountry* copy;
    Cint next_i = 0;
    Cint new_i = 1;
public:
    void add_el(const CCountry& CCountry);
    void remove_el(const int& index);
    void del_all();
    void get_to_Screen(const int& index) const;
    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&));
};

bool sortTitle(CCountry&, CCountry&);
bool sortNumber_of_cities(CCountry&, CCountry&);
bool sortPopulation(CCountry&, CCountry&);
bool sortArea(CCountry&, CCountry&);
bool sortCitizen(CCountry&, CCountry&);
bool sortTitle_2(CCountry&, CCountry&);
bool sortNumber_of_cities_2(CCountry&, CCountry&);
bool sortPopulation_2(CCountry&, CCountry&);
bool sortArea_2(CCountry&, CCountry&);
bool sortCitizen_2(CCountry&, CCountry&);
```

CMetod.cpp

```
#include "CMetod.h"
void CMetod::add_el(const CCountry& Country)
{
	if (next_i == 0)
	{
		countries = new CCountry[1];
		countries[next_i] = Country;
		next_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];
		}
		delete[] copy;
		countries[next_i] = Country;
		next_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)
        {
            delete[] countries;
            next_i = 0;
        }

}
void CMetod::get_to_Screen(const int& index) const
{
        std::cout << "Title " << "Number_of_cities " << "Population " << "Area " <<
"Place_of_birth_citizen " << "Birthday_citizen " << "\n";
        std::cout << get_str_by_file(index) << "\n";
}
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++)
        {
            get_to_Screen(i);
        }
}
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)
                get_to_Screen(i);
        }
}
std::string CMetod::get_str_by_file(const int& index) const
{
            std::stringstream ss;
            ss << "_" << countries[index].getTitle() << "_ " <<
countries[index].getNumber_of_cities() <<  " " << countries[index].getPopulation()<< " " <<
countries[index].getArea() << " " << countries[index].getPlace_of_birth_citizen() << " " <<
countries[index].getBirthday_citizen();
            return ss.str();
}
void CMetod::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;
                std::string Place_of_birth_citizen;
                s >> Place_of_birth_citizen;
                int i6 = str.find(" ", i5 + 1);
                temp = str.substr(i5 + 1, i6 - i5);
                s << temp;
                int Birthday_citizen;
                s >> Birthday_citizen;
                int i7 = str.find(" ", i6 + 1);
                temp = str.substr(i6 + 1, i7 - i6);
                s << temp;
                CCountry secondcountry(Title, Number_of_cities, Population,  Area,
Place_of_birth_citizen, Birthday_citizen);
                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-Яа-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-ZA-Я]");
	if (!std::regex_search(str, reg_5))
	{
		return false;
	}
	return true;
}
void CMetod::print_all_with_2_or_more_words() const
{
	for (int i = 0; i < next_i; i++)
	{
		std::string str;
		str = get_str_by_file(i);
		std::regex reg("_.+ .+_");
		if (std::regex_search(str, reg))
		{
			std::cout << i + 1 << " " << str << "\n";
		}
	}
}
void CMetod::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()));
}
bool sortTitle_2(CCountry& a, CCountry& b)
{
	return (a.getTitle() < b.getTitle());
}
bool sortNumber_of_cities_2(CCountry& a, CCountry& b)
{
	return(a.getNumber_of_cities() > b.getNumber_of_cities());
}
bool sortPopulation_2(CCountry& a, CCountry& b)
{
	return(a.getPopulation() > b.getPopulation());
}
bool sortArea_2(CCountry& a, CCountry& b)
{
	return (a.getArea() > b.getArea());
}
bool sortCitizen_2(CCountry& a, CCountry& b)
{
	return((a.getPlace_of_birth_citizen() < b.getPlace_of_birth_citizen()) &&
(a.getBirthday_citizen() > b.getBirthday_citizen()));
}
```

menu.h

```
#include "menu.h"
void menu()
{
    setlocale(LC_ALL, "Russian");
    int n = 0, temp_i;
    CMetod dir;
    CCountry firstcountry1("Страна1", 143, 45745656, 47342362, "Харьков", 22062012);
    dir.add_el(firstcountry1);
    CCountry firstcountry2("Страна2", 156, 38567454, 68457458, "Рим", 13012016);
    dir.add_el(firstcountry2);
    CCountry firstcountry3("Страна3", 167, 46357625, 98686453, "Ужгород", 31102007);
    dir.add_el(firstcountry3);
    CCountry firstcountry4("Страна4", 179, 78567583, 68457458, "Запорожье", 27072000);
    dir.add_el(firstcountry4);
    int c;
    int m;
    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)
        {
            CCountry firstcountry5("Страна5", 323, 93645665, 78767464, "Сумы", 24112001);
            dir.add_el(firstcountry5);
            std::cout << "Страна добавлена." << "\n";
        }
        else if (n == 2)
        {
            std::cout << "Введите номер удаляемого элемента (нумерация начинаеться с 1): ";
            std::cin >> temp_i;
            dir.remove_el(temp_i - 1);
            std::cout << "Страна удалена." << "\n";
        }
        else if (n == 3)
        {
            dir.print_all();
        }
        else if (n == 4)
        {
            dir.find_to_population_density();
        }
        else if (n == 5)
        {
            dir.write_to_file("text.txt");
        }
        else if (n == 6)
        {
            dir.read_from_file("text.txt");
        }
        else if (n == 7)
        {
            dir.print_all_with_2_or_more_words();
        }
        else if (n == 8)
        {
            std::cout << "Выберите порядок сортировки. 1 - по алфавиту и от большего к меньшему, 2 -
наоборот. " << "\n";
            std::cin >> m;
            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 && m == 1)
            {
                dir.sort(sortTitle);
            }
            else if (c == 1 && m == 2)
            {
                dir.sort(sortTitle_2);
            }
            else if (c == 2 && m == 1)
            {
                dir.sort(sortNumber_of_cities);
            }
            else if (c == 2 && m == 2)
            {
                dir.sort(sortNumber_of_cities_2);
            }
            else if (c == 3 && m == 1)
            {
                dir.sort(sortPopulation);
            }
            else if (c == 3 && m == 2)
            {
                dir.sort(sortPopulation_2);
            }
            else if (c == 4 && m == 1)
            {
```

```
                dir.sort(sortArea);
            }
            else if (c == 4 && m == 2)
            {
                dir.sort(sortArea_2);
            }
            else if (c == 5 && m == 1)
            {
                dir.sort(sortCitizen);
            }
            else if (c == 5 && m == 2)
            {
                dir.sort(sortCitizen_2);
            }           else
            {
                std::cout << "Неправильный номер." << "\n";
                n = 0;
                break;
            }
        }
    }
    dir.del_all();
}
```

tests.cpp

```
#pragma once
#include <iostream>
#include "menu.h"
#define _CRTDBG_MAP_ALLOC
int main()
{
	setlocale(LC_ALL, "Russian");
	CCountry firstcountry("Тест_страна1", 100, 11111111, 22222222, "New_York", 01012001);
	CCountry secondcountry("Тест_страна2", 100, 22222222, 33333333, "New_Vegas", 02022002);
	std::cout << "Тест будет пройден если сейчас на экран будет выведена следующая
последовательность: 0 0 1 1 1 - ";
	std::cout << sortTitle(firstcountry, secondcountry) << " " <<
sortNumber_of_cities(firstcountry, secondcountry) << " " << sortPopulation(firstcountry,
secondcountry) << " " << sortArea(firstcountry, secondcountry) << " " << sortCitizen(firstcountry,
secondcountry);
	std::cout << "\n";
	if (_CrtDumpMemoryLeaks())
	{
		std::cout << "Утечка памяти обнаружена." << "\n";
	}
	else
	{
		std::cout << "Утечка памяти не обнаружена." << "\n";
	}
	int t;
	std::cin >> t;
}
```

data.txt

```
_Страна1_ 143 45745656 47342362 Харьков 22062012
_Страна2_ 156 38567454 68457458 Рим 13012016
_Страна3_ 167 46357625 98686453 Ужгород 31102007
_Страна4_ 179 78567583 68457458 Запорожье 27072000
```

4. Результати роботи програми

Результати роботи програми:

```
Файл создан при помощи конструктора с аргументами.
Файл создан при помощи конструктора по умолчанию.
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 уничтожен при помощи деструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Файл создан при помощи конструктора по умолчанию.
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 уничтожен при помощи деструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
-_-_-_-_-_-_-_-_-_-_-_-_МЕНЮ-_-_-_-_-_-_-_-_-_-_-_-
-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-
_-_-_-_-_-_-_Выберите желаемую опцию:-_-_-_-_-_-_-_
-_-_-_-1 - добавить элемент в список.-_-_-_-_-_-_-_
-_-_-_-2 - удалить элемент из списка.-_-_-_-_-_-_-_
-_-_-_-3 - показать все элементы списка.-_-_-_-_-_-
-_-_-_-4 - найти наименьшую плотность населения страны.
-_-_-_-5 - записать данные а файл.-_-_-_-_-_-_-_-_-
-_-_-_-6 - считать данные из файла.-_-_-_-_-_-_-_-_
-_-_-_-7 - найти все элеметы в названии которых есть 2 или больше слова.
-_-_-_-8 - Отсортировать массив.-_-_-_-_-_-_-_-_-_-
-_-_-_-9 - завершить работу программы.-_-_-_-_-_-_-
-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-
9
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.
Файл уничтожен при помощи деструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Файл уничтожен при помощи деструктора по умолчанию.
Утечка памяти не обнаружена.
```

Результати тестів:

```
Файл создан при помощи конструктора с аргументами.
Файл создан при помощи конструктора с аргументами.
Тест будет пройден если сейчас на экран будет выведена следующая последовательность: 0 0 1 1 1 - 0 0 1 1 1
Утечка памяти не обнаружена.
```

5. Висновки

При виконанні даної лабораторної роботи було набуто практичного досвіду роботи з агрегацією та композицією.

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