

Звіт

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

Лабораторна робота №16. РОБОТА З ДИНАМІЧНОЮ ПАМ'ЯТТЮ

Тема. Системна робота з динамічною пам'яттю.

Мета: дослідити особливості мови C++ при роботі з динамічною пам'яттю.

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

Маючи класи з прикладної області РЗ (тільки базовий клас та клас / класи-спадкоємці), перевантажити оператори new / new [] та delete / delete [].

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 повинен демонструвати композицію: CCitizen

2.2 Опис змінних

`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).

`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`.

`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

`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.

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

3 Текст програми

Лабораторная работа 16.cpp

```
#include "file.h"
#define _CRTDBG_MAP_ALLOC

int main(){
    setlocale(LC_ALL, "Russian");
    file* test_ptr;
    test_ptr = new file;
    delete test_ptr;
    test_ptr = new file[2];
    delete[] test_ptr;
    executable_file* test_ptr2;
    test_ptr2 = new executable_file;
    delete test_ptr2;
    test_ptr2 = new executable_file[2];
    delete[] test_ptr2;
    non_executable_file* test_ptr3;
    test_ptr3 = new non_executable_file;
    delete test_ptr3;
```

```

test_ptr3 = new non_executable_file[2];
delete[] test_ptr3;
if (_CrtDumpMemoryLeaks()) {
    std::cout << "Утечка памяти обнаружена." << "\n";
}
else {
    std::cout << "Утечка памяти не обнаружена." << "\n";
}
}

```

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);
        void* operator new(size_t);
        void* operator new[](size_t);
        void operator delete(void*);
        void operator delete[](void*);
};
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);
    void* operator new(size_t);
    void* operator new[](size_t);
    void operator delete(void*);
    void operator delete[](void*);

};
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);
    void* operator new(size_t);
    void* operator new[](size_t);
    void operator delete(void*);
    void operator delete[](void*);

};
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);

```

file.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();
    }
}

void* CCountry::operator new(size_t data) {
    std::cout << "Тут сработал оператор new класса CCountry.\n";
    return ::operator new(data);
}

```



```

}
void* CCountry::operator new[](size_t data) {
    std::cout << "Тут сработал оператор new[] класса CCountry.\n";
    return ::operator new[](data);
}
void CCountry::operator delete(void* ptr) {
    std::cout << "Тут сработал оператор delete класса CCountry.\n";
    return ::operator delete(ptr);
}
void CCountry::operator delete[](void* ptr) {
    std::cout << "Тут сработал оператор delete[] класса CCountry.\n";
    return ::operator delete[](ptr);
}
void* Inheritor_CCountry::operator new(size_t data) {
    std::cout << "Тут сработал оператор new класса Inheritor_CCountry.\n";
    return ::operator new(data);
}
void* Inheritor_CCountry::operator new[](size_t data) {
    std::cout << "Тут сработал оператор new[] класса Inheritor_CCountry.\n";
    return ::operator new[](data);
}
void Inheritor_CCountry::operator delete(void* ptr) {
    std::cout << "Тут сработал оператор delete класса Inheritor_CCountry.\n";
    return ::operator delete(ptr);
}
void Inheritor_CCountry::operator delete[](void* ptr) {
    std::cout << "Тут сработал оператор delete[] класса Inheritor_CCountry.\n";
    return ::operator delete[](ptr);
}
void* Inheritor_CCountry_second::operator new(size_t data) {
    std::cout << "Тут сработал оператор new класса Inheritor_CCountry_second.\n";
    return ::operator new(data);
}
void* Inheritor_CCountry_second::operator new[](size_t data) {
    std::cout << "Тут сработал оператор new[] класса Inheritor_CCountry_second.\n";
    return ::operator new[](data);
}
void Inheritor_CCountry_second::operator delete(void* ptr) {
    std::cout << "Тут сработал оператор delete класса Inheritor_CCountry_second.\n";
    return ::operator delete(ptr);
}
void Inheritor_CCountry_second::operator delete[](void* ptr) {
    std::cout << "Тут сработал оператор delete[] класса Inheritor_CCountry_second.\n";
    return ::operator delete[](ptr);
}
}

```

4. Результати роботи програми

Результати роботи програми:

```
Файл создан при помощи конструктора по умолчанию.  
Файл уничтожен при помощи деструктора по умолчанию.  
Файл уничтожен при помощи деструктора по умолчанию.  
Тут сработал оператор delete[] класса CCountry.  
Тут сработал оператор new класса Inheritor_CCountry.  
Файл создан при помощи конструктора по умолчанию.  
Файл уничтожен при помощи деструктора по умолчанию.  
Тут сработал оператор delete класса Inheritor_CCountry.  
Тут сработал оператор new[] класса Inheritor_CCountry.  
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.  
Тут сработал оператор delete[] класса Inheritor_CCountry.  
Тут сработал оператор new класса Inheritor_CCountry.  
Файл создан при помощи конструктора по умолчанию.  
Файл уничтожен при помощи деструктора по умолчанию.  
Тут сработал оператор delete класса Inheritor_CCountry.  
Тут сработал оператор new[] класса Inheritor_CCountry.  
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.  
Тут сработал оператор delete[] класса Inheritor_CCountry.  
Утечка памяти не обнаружена.
```

5. Висновки

При виконанні даної лабораторної роботи було використано переважано методи new,new[],delete і delete[], вони були переважені в тілі самого класа.

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