МІНІСТЕРСТВО ОСВІТИ І НАУКИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ «ХПІ»

Кафедра «Обчислювальна техніка та програмування»

Розрахункове завдання з програмування

Тема: «РОЗРОБКА ІНФОРМАЦІЙНО-ДОВІДКОВОЇ СИСТЕМИ»

Пояснювальна записка

КІТ.119а.

Розробники
Виконав:
студент групи КІТ-119а
_____/ Капелька Я.І./

Перевірив:
_____/аспірант Бартош М. В./

Харків 2020

ЗАТВЕРДЖЕНО

КІТ.119а.

Розрахункове завдання з програмування

Тема: «РОЗРОБКА ІНФОРМАЦІЙНО-ДОВІДКОВОЇ СИСТЕМИ»

Пояснювальна записка

КІТ.119а.

Аркушів _27_

Харків 2020

ЗМІСТ

ВСТУП

Поняття «Інформаційна система»

Інформаційно-довідкові системи – це сукупність організаційних і технічних засобів, що призначені для керування базами даних і використовуються, наприклад, для ведення статистики, складання каталогів тощо. Вони полегшують оперування великими об’ємами професійно цінної інформації, виступаючи як засіб надійного збереження професійних знань, забезпечуючи зручний і швидкий пошук необхідних відомостей.

Призначення та галузь застосування

Призначення розробки – оперування даними про прикладну галузь література, а саме про підручники. Розроблена з використанням ієрархії класів програма дозволяє виконувати такі завдання: читання даних з файлу та їх запис у контейнер, запис даних з контейнера у файл, сортування елементів у контейнері за вказаними критеріями (поле та напрям задаються користувачем з клавіатури), виконання особистого завдання. Також було розроблено декілька інших класів, які слугують для: відображення діалогового меню, тестування розроблених методів класу, сортування.

Постановка завдання до розробки

В основі функціонування інформаційно-довідкових систем лежить обробка інформації. Режими її обробки можуть бути такими: пакетний, діалоговий, реального часу.

Пакетний режим визначає операції та їх послідовність з формування даних в ЕОМ і формування розрахунків безпосередньо на обчислювальному центрі чи відповідною системою.

Діалоговий режим забезпечує безпосередню взаємодію користувача з системою. Ініціатором діалогу може бути як користувач, так і ЕОМ. В останньому випадку на кожному кроці користувачу повідомляється, що треба робити.

Режим реального часу — режим обробки інформації системою при взаємодії з зовнішніми процесами в темпі ходу цих процесів.

В роботі буде реалізовано діалоговий режим обробки інформації, де ініціатором виступає ЕОМ.

Дані, що обробляються, в оперативній пам’яті можуть зберігатися у вигляді масиву або лінійного (одно- або двонаправленого) списку.

До переваг масиву можна віднести:

1. Ефективність при звертанні до довільного елементу, яке відбувається за постійний час O(1),
2. Можливість компактного збереження послідовності їх елементів в локальній області пам'яті, що дозволяє ефективно виконувати операції з послідовного обходу елементів таких масивів.
3. Масиви є дуже економною щодо пам'яті структурою даних.

До недоліків:

1. Операції, такі як додавання та видалення елементу, потребують часу O(n), де n — розмір масиву.
2. У випадках, коли розмір масиву є досить великий, використання звичайного звертання за індексом стає проблематичним.
3. Масиви переважно потребують непреривної області для зберігання.

До переваг списку можна віднести:

1. Списки досить ефективні щодо операцій додавання або видалення елементу в довільному місці списку, виконуючи їх за постійний час.
2. В списках також не існує проблеми «розширення», яка рано чи пізно виникає в масивах фіксованого розміру, коли виникає необхідність включити в нього додаткові елементи.
3. Функціонування списків можливо в ситуації, коли пам'ять комп'ютера фрагментована.

До недоліків:

1. Для доступу до довільного елементу необхідно пройти усі елементи перед ним.
2. Необхідність разом з корисною інформацією додаткового збереження інформації про вказівники, що позначається на ефективності використання пам'яті цими структурами.

Виходячи з переваг та недоліків зазначених вище в розроблюваній програмі для подання даних буде реалізовано вектор, який є абстрактною моделлю, що імітує динамічний масив.

Для реалізації поставленого завдання було обрано об’єктно-орієнтовану мову програмування C++, через те, що вона засновує програми як сукупності взаємодіючих об'єктів, кожен з яких є екземпляром певного класу, а класи є членами певної ієрархії наслідування. А середовищем програмування – Microsoft Visual Studio.

РОЗРОБКА ІНФОРМАЦІЙНО-ДОВІДКОВОЇ СИСТЕМИ

Розробка алгоритмів програми

При розробленні структур даних було створено: базовий клас CCountry, який наслідують класи Inheritor_CCountry та Inheritor_CCountry_second. На рис. 1 показано внутрішню структуру, а на рис. 2 - відносини розроблених класів у вигляді UML-діаграми.

а) б) в)

Рисунок 1 – Поля базового класу (а), а також класів-спадкоємців (б, в)

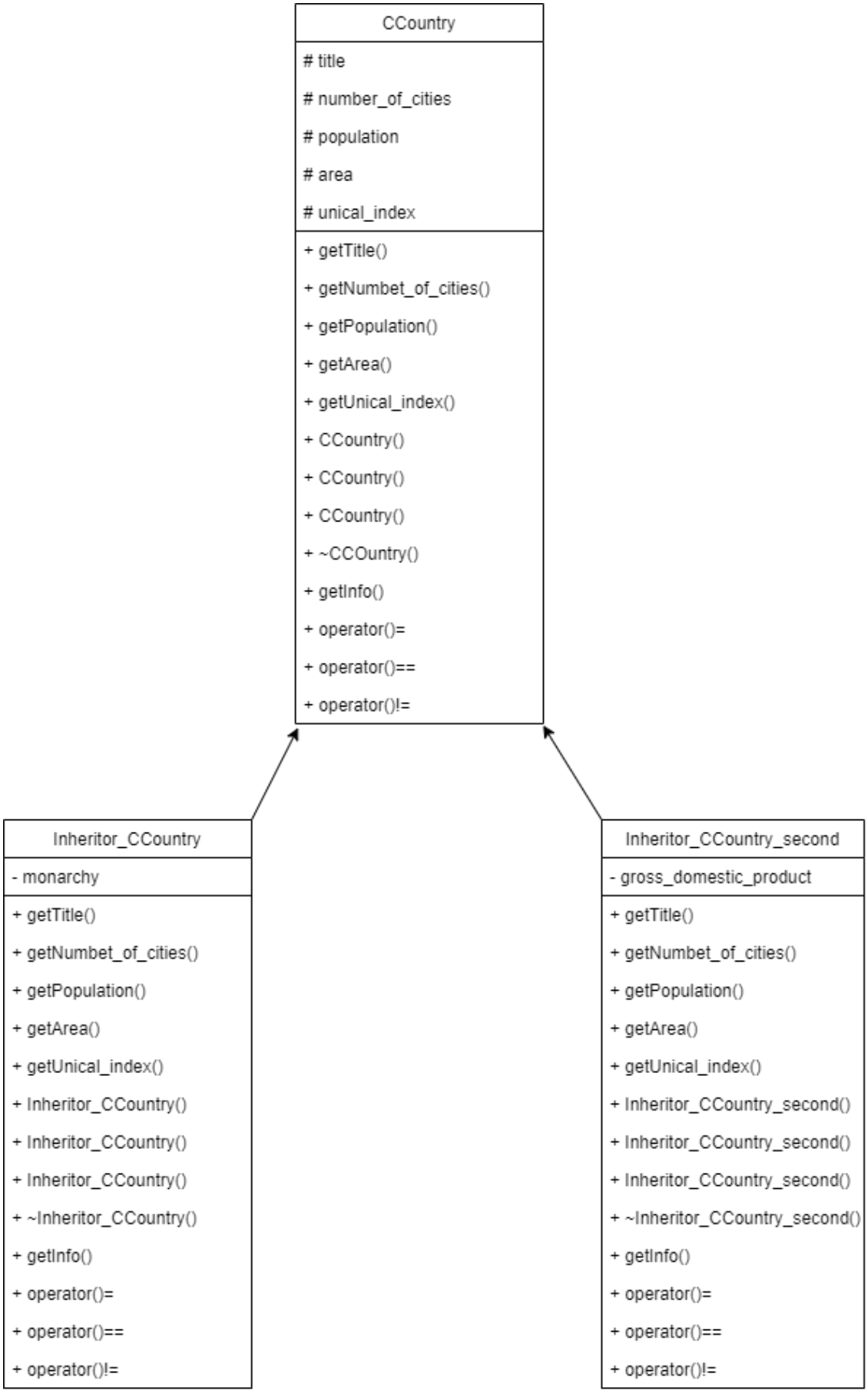


Рисунок 2 – Схема ієрархії розроблених класів

Дані про підручники будуть заноситися до списку. Для цього було розроблено клас-контролер CMetod з полями показаними на рис. 3 і методами на рис. 4.

Приватні дані
CCountry **countries
CCountry **copy
Cint next_i = 0;
Cint new_i = 1;

Рисунок 3 – Поля класу-контролеру CMetod

Приватні дані

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();
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&));
CCountry* operator[](const int& index):
friend std::ostream& operator<< (std::ostream& os, CMetod& Cmetod);

Рисунок 4 – Розроблені методи класу CMetod

На рис. 5 подано структуру проекту розробленого програмного продукту.

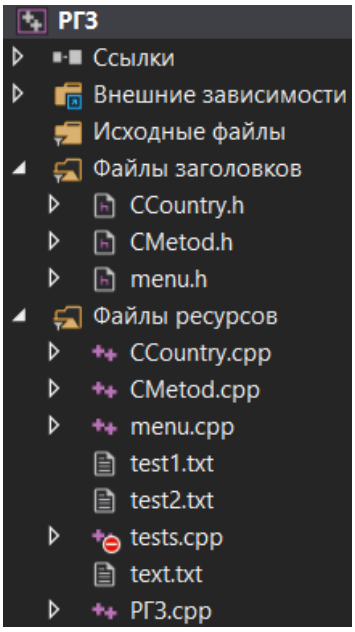


Рисунок 5 – Структура проекту

СХЕМИ АЛГОРИТМУ ПРОГРАМИ

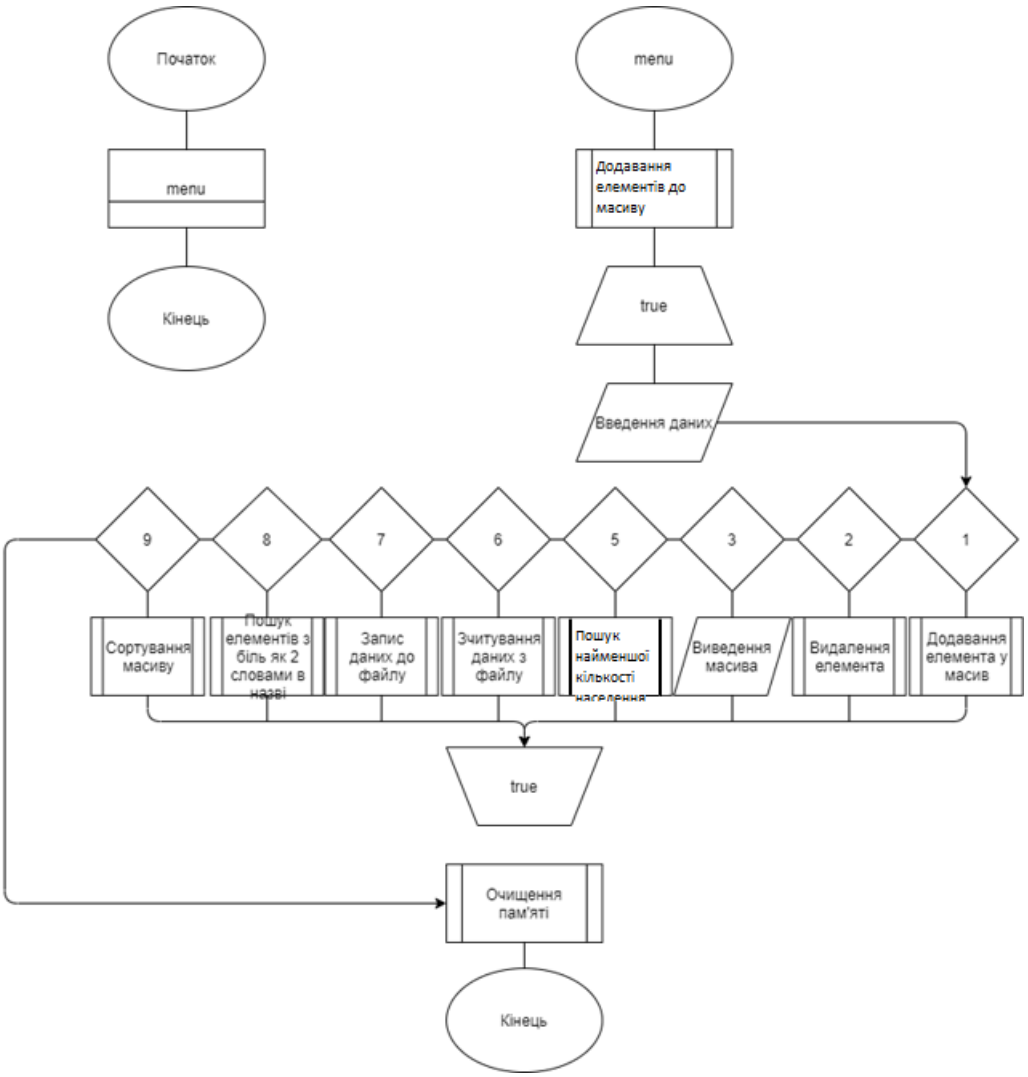


Рисунок 1 – Схема алгоритму функції menu

ВИСНОВОК

У результаті розробки інформаційно-довідкової системи було виконано наступні завдання:

1. Досліджено літературу стосовно прикладної галузі та оформлено аналітичний розділ пояснювальної записки;
2. Для прикладної галузі файл розроблено розгалужену ієрархію класів, що складається з трьох класів – один «батьківський», два спадкоємці. У них було перевантажено оператори введення-виведення та оператор порівняння;
3. Розроблено клас-контролер, що включає масив базового класу, та класів спадкоємців.
4. Оформлено схеми алгоритмів функцій класів контролера та діалогового меню;
5. Оформлено документацію;
6. Було додано обробку помилок, перевірку вхідних даних за допомогою регулярних виразів;

СПИСОК ДЖЕРЕЛ ІНФОРМАЦІЇ

1. Дейтел Х.М. Как программировать на Си++ / Х.М. Дейтел, П.Дж. Дейтел М. : ЗАО БИНОМ, 1999. – 1000 с.
2. Штейн Клифорд (2019). Алгоритмы. Построение и анализ.
3. Вандервуд, Джосаттис - Шаблоны C++. Справочник разработчика. / Пер. с англ. - М.: Вильямс, 2008. – 536 с.
4. Андрей Александреску, Современное проектирование на C++.М.:ООО «И.Д.Вильямс», 2002.
5. Страуструп Б. Дизайн и эволюция C++ / Б. Страуструп; пер. с англ. – М. : ДМК Пресс; С.Пб: Питер, 2007. – 445 с.

6. Остерн Обобщенное программирование и STL: Использование инаращивание стандартной библиотеки шаблонов C++ / Остерн; Пер. сангл. – С.Пб: Невский Диалект, 2004. – 544 с.

Додаток А

Текст програми

`#include <CCountry.h>`

Inherited by **Inheritor_??ountry**, and **Inheritor_??ountry_second**.

Public Member Functions

	CCountry ()
	CCountry (const **CCountry** &)
	CCountry (const std::string &, const int &, const int &, const int &, const bool &, const int &)
	~CCountry ()
virtual bool	**getPlace_of_birth_citizen** () const
virtual **Cint**	**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 int &)
virtual bool	**getMonarchy** () const
virtual std::string	**getInfo** () const =0

CCountry &	operator= (const CCountry &Country)

Public Attributes

Cint	type_of_Country = 0

Protected Attributes

std::string	title
Cint	population_density
Cint	number_of_cities
Cint	population
Cint	area
Cint	unical_index
C□itizen	citizen

Friends

bool	operator== (const CCountry &Country1, const CCountry &Country2)
bool	operator!= (const CCountry &Country1, const CCountry &Country2)

Detailed Description

Definition at line 20 of file CCountry.h.

Constructor & Destructor Documentation

◆ CCountry() [1/3]

CCountry::CCountry ( )

Definition at line 18 of file CCountry.cpp.

◆ CCountry() [2/3]

CCountry::CCountry (const CCountry & CCountry)

Definition at line 30 of file CCountry.cpp.

◆ CCountry() [3/3]

```
CCountry::CCountry ( const std::string &  Title,
                     const int &          Number_of_cities,
                     const int &          Population,
                     const int &          Area,
                     const bool &         Place_of_birth_citizen,
                     const int &          Birthday_citizen
                   )
```

Definition at line **40** of file **CCountry.cpp**.

◆ ~CCountry()

```
CCountry::~CCountry ( )
```

Definition at line **51** of file **CCountry.cpp**.

Member Function Documentation

◆ getArea()

```
int CCountry::getArea ( ) const
```
virtual

Definition at line **6** of file **CCountry.cpp**.

◆ getBirthday_citizen()

```
int CCountry::getBirthday_citizen ( ) const
```
virtual

Definition at line **9** of file **CCountry.cpp**.

◆ getInfo()

```
virtual std::string CCountry::getInfo ( ) const
```
pure virtual

Implemented in **Inheritor_��ountry_second**, and **Inheritor_��ountry**.

◆ getMonarchy()

```
bool CCountry::getMonarchy ( ) const
```
virtual

Reimplemented in **Inheritor_��ountry**.

Definition at line **59** of file **CCountry.cpp**.

◆ getNumber_of_cities()

```
int CCountry::getNumber_of_cities ( ) const
```
virtual

Definition at line **4** of file **CCountry.cpp**.

◆ getPlace_of_birth_citizen()

```
bool CCountry::getPlace_of_birth_citizen ( ) const
```
virtual

Definition at line **8** of file **CCountry.cpp**.

◆ getPopulation()

```
int CCountry::getPopulation ( ) const
```
virtual

Definition at line **5** of file **CCountry.cpp**.

◆ getPopulation_density()

```
int CCountry::getPopulation_density ( ) const
```
virtual

Definition at line **3** of file **CCountry.cpp**.

◆ getTitle()

```
std::string CCountry::getTitle ( ) const
```
virtual

Definition at line **2** of file **CCountry.cpp**.

◆ getUnical_index()

```
int CCountry::getUnical_index ( ) const
```

virtual

Definition at line 7 of file CCountry.cpp.

◆ operator=()

```
CCountry& CCountry::operator= ( const CCountry &  Country )
```

inline

Definition at line 54 of file CCountry.h.

◆ setArea()

```
void CCountry::setArea ( const int &  Area )
```

virtual

Definition at line 14 of file CCountry.cpp.

◆ setBirthday_citizen()

```
void CCountry::setBirthday_citizen ( const int &  Birthday_citizen )
```

virtual

Definition at line 17 of file CCountry.cpp.

◆ setNumber_of_cities()

```
void CCountry::setNumber_of_cities ( const int &  Number_of_cities )
```

virtual

Definition at line 12 of file CCountry.cpp.

◆ setPlace_of_birth_citizen()

```
void CCountry::setPlace_of_birth_citizen ( const bool &  Place_of_birth_citizen )
```

virtual

Definition at line 16 of file CCountry.cpp.

◆ setPopulation()

```
void CCountry::setPopulation ( const int &  Population )
```

virtual

Definition at line 13 of file CCountry.cpp.

◆ setPopulation_density()

```
void CCountry::setPopulation_density ( const int &  Population_density )
```

virtual

Definition at line 11 of file CCountry.cpp.

◆ setTitle()

```
void CCountry::setTitle ( const std::string &  Title )
```

virtual

Definition at line 10 of file CCountry.cpp.

◆ setUnical_index()

```
void CCountry::setUnical_index ( const int &  Unical_index )
```

virtual

Definition at line 15 of file CCountry.cpp.

Friends And Related Function Documentation

◆ operator!=

```
bool operator!= ( const CCountry &  Country1,
                  const CCountry &  Country2
                )
```

friend

Definition at line 116 of file CCountry.cpp.

◆ operator!=

```
bool operator!= ( const CCountry &  Country1,
                  const CCountry &  Country2
                )
```

friend

Definition at line 116 of file CCountry.cpp.

◆ operator==

```
bool operator== ( const CCountry &  Country1,
                  const CCountry &  Country2
                )
```

friend

Definition at line 85 of file CCountry.cpp.

Member Data Documentation

◆ area

Cint CCountry::area

protected

Definition at line 27 of file CCountry.h.

◆ citizen

C◆itizen CCountry::citizen

protected

Definition at line 29 of file CCountry.h.

◆ number_of_cities

Cint CCountry::number_of_cities

protected

Definition at line 25 of file CCountry.h.

◆ population

Cint CCountry::population

protected

Definition at line 26 of file CCountry.h.

◆ population_density

Cint CCountry::population_density

protected

Definition at line 24 of file CCountry.h.

◆ title

std::string CCountry::title

protected

Definition at line 23 of file CCountry.h.

◆ type_of_Country

Cint CCountry::type_of_Country = 0

Definition at line 31 of file CCountry.h.

◆ unical_index

Cint CCountry::unical_index

protected

Definition at line 28 of file CCountry.h.

CMetod Class Reference

`#include <CMetod.h>`

Public Member Functions

void	**add_el** (const **Inheritor_��ountry** &Inheritor_ountry)
void	**add_el** (const **Inheritor_��ountry_second** &Inheritor_ountry_second)
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** &))
CCountry *	**operator[]** (const int &index)

Friends

std::ostream &	**operator<<** (std::ostream &os, **CMetod** &Cmetod)

Detailed Description

Definition at line 3 of file **CMetod.h**.

Member Function Documentation

◆ add_el() [1/2]

void CMetod::add_el (const **Inheritor_��ountry** & Inheritor_�ountry)

Definition at line 4 of file **CMetod.cpp**.

◆ add_el() [2/2]

void CMetod::add_el (const **Inheritor_��ountry_second** & Inheritor_�ountry_second)

Definition at line 32 of file **CMetod.cpp**.

◆ check_str()

bool CMetod::check_str (const std::string & str) const

Definition at line 239 of file **CMetod.cpp**.

◆ del_all()

void CMetod::del_all ( )

Definition at line **88** of file **CMetod.cpp**.

◆ find_to_index()

CCountry * CMetod::find_to_index (const int & index) const

Definition at line **106** of file **CMetod.cpp**.

◆ find_to_population_density()

void CMetod::find_to_population_density ( ) const

Definition at line **123** of file **CMetod.cpp**.

◆ find_to_str_by_file()

void CMetod::find_to_str_by_file (const std::string str)

Definition at line **145** of file **CMetod.cpp**.

◆ get_str_by_file()

std::string CMetod::get_str_by_file (const int & index) const

Definition at line **139** of file **CMetod.cpp**.

◆ get_to_Screen()

void CMetod::get_to_Screen (const int & index) const

Definition at line **101** of file **CMetod.cpp**.

◆ operator[]()

CCountry* CMetod::operator[] (const int & index) inline

Definition at line **27** of file **CMetod.h**.

◆ print_all()

void CMetod::print_all ( ) const

Definition at line **116** of file **CMetod.cpp**.

◆ print_all_with_2_or_more_words()

void CMetod::print_all_with_2_or_more_words ( ) const

Definition at line **268** of file **CMetod.cpp**.

◆ read_from_file()

void CMetod::read_from_file (const std::string name)

Definition at line **225** of file **CMetod.cpp**.

◆ remove_el()

void CMetod::remove_el (const int & index)

Definition at line **60** of file **CMetod.cpp**.

◆ sort()

void CMetod::sort (bool(*)(**CCountry** &, **CCountry** &) comp)

Definition at line **281** of file **CMetod.cpp**.

◆ write_to_file()

void CMetod::write_to_file (const std::string name)

Definition at line **210** of file **CMetod.cpp**.

Friends And Related Function Documentation

◆ operator<<

std::ostream& operator<< (std::ostream & os,
CMetod & Cmetod
)

friend

Definition at line **340** of file **CMetod.cpp**.

C◆itizen Class Reference

`#include <CCountry.h>`

Public Member Functions

const bool	**getPlace_of_birth_citizen** () const
Cint	**getBirthday_citizen** () const
void	**setPlace_of_birth_citizen** (const bool &)
void	**setBirthday_citizen** (const int &)

Detailed Description

Definition at line **9** of file **CCountry.h**.

Member Function Documentation

◆ getBirthday_citizen()

Cint C◆itizen::getBirthday_citizen ( ) const

Definition at line **56** of file **CCountry.cpp**.

◆ getPlace_of_birth_citizen()

const bool C�itizen::getPlace_of_birth_citizen ( ) const

Definition at line 55 of file CCountry.cpp.

◆ setBirthday_citizen()

void C�itizen::setBirthday_citizen (const int & Birthday_citizen)

Definition at line 58 of file CCountry.cpp.

◆ setPlace_of_birth_citizen()

void C�itizen::setPlace_of_birth_citizen (const bool & Place_of_birth_citizen)

Definition at line 57 of file CCountry.cpp.

Inheritor_��ountry Class Reference final

#include <CCountry.h>

Inherits CCountry.

Public Member Functions

virtual bool	getMonarchy () const override final
virtual void	setMonarchy (const bool &) final
	Inheritor_��ountry ()
	Inheritor_��ountry (const Inheritor_��ountry &)
	Inheritor_��ountry (const std::string &, const int &, const int &, const int &, const bool &, const int &, const bool &)
	~Inheritor_��ountry ()
virtual std::string	getInfo () const final
Inheritor_��ountry &	operator= (const Inheritor_��ountry &Inheritor_�ountry)

- Public Member Functions inherited from CCountry

Friends

bool	operator== (const Inheritor_��ountry &Inheritor_�ountry1, const Inheritor_��ountry &Inheritor_�ountry2)
bool	operator!= (const Inheritor_��ountry &Inheritor_�ountry1, const Inheritor_��ountry &Inheritor_�ountry2)

Additional Inherited Members

- Public Attributes inherited from CCountry

Cint	type_of_Country = 0

- Protected Attributes inherited from CCountry

std::string	title
Cint	population_density
Cint	number_of_cities
Cint	population
Cint	area
Cint	unical_index
C□itizen	citizen

Detailed Description

Definition at line 68 of file CCountry.h.

Constructor & Destructor Documentation

◆ Inheritor_□□ountry() [1/3]

Inheritor_□□ountry::Inheritor_□□ountry ( )

Definition at line 68 of file CCountry.cpp.

◆ Inheritor_��ountry() [2/3]

Inheritor_��ountry::Inheritor_��ountry (const **Inheritor_��ountry** & in_CC)

Definition at line **69** of file **CCountry.cpp**.

◆ Inheritor_��ountry() [3/3]

```
Inheritor_��ountry::Inheritor_��ountry ( const std::string &  Title,
                                         const int &          Number_of_cities,
                                         const int &          Population,
                                         const int &          Area,
                                         const bool &         Place_of_birth_citizen,
                                         const int &          Birthday_citizen,
                                         const bool &         Monarchy
                                       )
```

Definition at line **70** of file **CCountry.cpp**.

◆ ~Inheritor_��ountry()

Inheritor_��ountry::~Inheritor_��ountry ( )

Definition at line **71** of file **CCountry.cpp**.

Member Function Documentation

◆ getInfo()

std::string Inheritor_��ountry::getInfo ( ) const final virtual

Implements **CCountry**.

Definition at line **62** of file **CCountry.cpp**.

◆ getMonarchy()

bool Inheritor_��ountry::getMonarchy ( ) const final override virtual

Reimplemented from **CCountry**.

Definition at line **60** of file **CCountry.cpp**.

◆ operator=()

Inheritor_��ountry& Inheritor_��ountry::operator= (const **Inheritor_��ountry** & Inheritor_�ountry) inline

Definition at line **80** of file **CCountry.h**.

◆ setMonarchy()

void Inheritor_��ountry::setMonarchy (const bool & Monarchy) final virtual

Definition at line 61 of file CCountry.cpp.

Friends And Related Function Documentation

◆ operator!=

```
bool operator!= ( const Inheritor_��ountry &  Inheritor_�ountry1,
                  const Inheritor_��ountry &  Inheritor_�ountry2
                )
```
friend

Definition at line 155 of file CCountry.cpp.

◆ operator==

```
bool operator== ( const Inheritor_��ountry &  Inheritor_�ountry1,
                  const Inheritor_��ountry &  Inheritor_�ountry2
                )
```
friend

Definition at line 120 of file CCountry.cpp.

Inheritor_��ountry_second Class Reference final

`#include <CCountry.h>`

Inherits **CCountry**.

Public Member Functions

virtual bool	**getGross_domestic_product** () const final
virtual void	**setGross_domestic_product** (const bool &) final
	Inheritor_��ountry_second ()
	Inheritor_��ountry_second (const **Inheritor_��ountry_second** &)
	Inheritor_��ountry_second (const std::string &, const int &, const int &, const int &, const bool &, const int &, const bool &)
	~Inheritor_��ountry_second ()
virtual std::string	**getInfo** () const final
Inheritor_��ountry_second &	**operator=** (const **Inheritor_��ountry_second** &Inheritor_�ountry_second)

- Public Member Functions inherited from **CCountry**

Friends

bool	**operator==** (const **Inheritor_��ountry_second** &Inheritor_�ountry_second1, const **Inheritor_��ountry_second** &Inheritor_�ountry_second2)
bool	**operator!=** (const **Inheritor_��ountry_second** &Inheritor_�ountry_second1, const **Inheritor_��ountry_second** &Inheritor_�ountry_second2)

Additional Inherited Members

- Public Attributes inherited from CCountry

Cint	type_of_Country = 0

- Protected Attributes inherited from CCountry

std::string	title
Cint	population_density
Cint	number_of_cities
Cint	population
Cint	area
Cint	unical_index
C&#65533;itizen	citizen

Detailed Description

Definition at line 97 of file CCountry.h.

Constructor & Destructor Documentation

◆ Inheritor_&#65533;&#65533;ountry_second() [1/3]

Inheritor_&#65533;&#65533;ountry_second::Inheritor_&#65533;&#65533;ountry_second ( )

Definition at line 75 of file CCountry.cpp.

◆ Inheritor_��ountry_second() [2/3]

Inheritor_��ountry_second::Inheritor_��ountry_second (const **Inheritor_��ountry_second** & in_CC_second)

Definition at line **76** of file **CCountry.cpp**.

◆ Inheritor_��ountry_second() [3/3]

```
Inheritor_��ountry_second::Inheritor_��ountry_second ( const std::string &  Title,
                                                       const int &          Number_of_cities,
                                                       const int &          Population,
                                                       const int &          Area,
                                                       const bool &         Place_of_birth_citizen,
                                                       const int &          Birthday_citizen,
                                                       const bool &         Gross_domestic_product
                                                     )
```

Definition at line **77** of file **CCountry.cpp**.

◆ ~Inheritor_��ountry_second()

Inheritor_��ountry_second::~Inheritor_��ountry_second ( )

Definition at line **78** of file **CCountry.cpp**.

Member Function Documentation

◆ getGross_domestic_product()

bool Inheritor_��ountry_second::getGross_domestic_product ( ) const final virtual

Definition at line **73** of file **CCountry.cpp**.

◆ getInfo()

std::string Inheritor_��ountry_second::getInfo ( ) const final virtual

Implements **CCountry**.

Definition at line **79** of file **CCountry.cpp**.

◆ operator=()

Inheritor_��ountry_second& Inheritor_��ountry_second::operator= (const **Inheritor_��ountry_second** & Inheritor_�ountry_second) inline

Definition at line **109** of file **CCountry.h**.

◆ setGross_domestic_product()

void Inheritor_��ountry_second::setGross_domestic_product (const bool & Gross_domestic_product) final virtual

Definition at line 74 of file CCountry.cpp.

Friends And Related Function Documentation

◆ operator!=

bool operator!= (const Inheritor_��ountry_second & Inheritor_�ountry_second1,
const Inheritor_��ountry_second & Inheritor_�ountry_second2
) friend

Definition at line 194 of file CCountry.cpp.

◆ operator==

bool operator== (const Inheritor_��ountry_second & Inheritor_�ountry_second1,
const Inheritor_��ountry_second & Inheritor_�ountry_second2
) friend

Definition at line 159 of file CCountry.cpp.

CCountry.cpp File Reference

```
#include "CCountry.h"
```

Go to the source code of this file.

Functions

bool	**operator==** (const **CCountry** &Country1, const **CCountry** &Country2)
bool	**operator!=** (const **CCountry** &Country1, const **CCountry** &Country2)
bool	**operator==** (const **Inheritor_��ountry** &Inheritor_�ountry1, const **Inheritor_��ountry** &Inheritor_�ountry2)
bool	**operator!=** (const **Inheritor_��ountry** &Inheritor_�ountry1, const **Inheritor_��ountry** &Inheritor_�ountry2)
bool	**operator==** (const **Inheritor_��ountry_second** &Inheritor_�ountry_second1, const **Inheritor_��ountry_second** &Inheritor_�ountry_second2)
bool	**operator!=** (const **Inheritor_��ountry_second** &Inheritor_�ountry_second1, const **Inheritor_��ountry_second** &Inheritor_�ountry_second2)
bool	**check_str** (const std::string &str)
std::ostream &	**operator<<** (std::ostream &os, const **CCountry** &Country)
std::ostream &	**operator<<** (std::ostream &os, const **Inheritor_��ountry** &Inheritor_�ountry)
std::ostream &	**operator<<** (std::ostream &os, const **Inheritor_��ountry_second** &Inheritor_�ountry_second)
std::istream &	**operator>>** (std::istream &is, **Inheritor_��ountry** &Inheritor_�ountry)
std::istream &	**operator>>** (std::istream &is, **Inheritor_��ountry_second** &Inheritor_�ountry_second)

Function Documentation

◆ check_str()

```
bool check_str ( const std::string &  str )
```

Definition at line **198** of file **CCountry.cpp**.

◆ operator!=() [1/3]

```
bool operator!= ( const CCountry &  Country1,
                  const CCountry &  Country2
                )
```

Definition at line **116** of file **CCountry.cpp**.

◆ operator!=() [2/3]

```
bool operator!= ( const Inheritor_��ountry &  Inheritor_�ountry1,
                  const Inheritor_��ountry &  Inheritor_�ountry2
                )
```

Definition at line **155** of file **CCountry.cpp**.

◆ operator!=() [3/3]

```
bool operator!= ( const Inheritor_��ountry_second &  Inheritor_�ountry_second1,
                  const Inheritor_��ountry_second &  Inheritor_�ountry_second2
                )
```

Definition at line **194** of file **CCountry.cpp**.

◆ operator<<() [1/3]

```
std::ostream& operator<< ( std::ostream &       os,
                           const CCountry &     Country
                         )
```

Definition at line **222** of file **CCountry.cpp**.

◆ operator<<() [2/3]

```
std::ostream& operator<< ( std::ostream &                os,
                           const Inheritor_��ountry &    Inheritor_�ountry
                         )
```

Definition at line **226** of file **CCountry.cpp**.

◆ operator<<() [3/3]

```
std::ostream& operator<< ( std::ostream &                    os,
                           const Inheritor_��ountry_second &  Inheritor_�ountry_second
                         )
```

Definition at line **230** of file **CCountry.cpp**.

◆ operator==() [1/3]

```
bool operator== ( const CCountry &  Country1,
                  const CCountry &  Country2
                )
```

Definition at line **85** of file **CCountry.cpp**.

◆ operator==() [2/3]

```
bool operator== ( const Inheritor_��ountry &  Inheritor_�ountry1,
                  const Inheritor_��ountry &  Inheritor_�ountry2
                )
```

Definition at line **120** of file **CCountry.cpp**.

◆ operator==() [3/3]

```
bool operator== ( const Inheritor_��ountry_second &  Inheritor_�ountry_second1,
                  const Inheritor_��ountry_second &  Inheritor_�ountry_second2
                )
```

Definition at line **159** of file **CCountry.cpp**.

◆ operator>>() [1/2]

```
std::istream& operator>> ( std::istream &              is,
                           Inheritor_��ountry &  Inheritor_�ountry
                         )
```

Definition at line **234** of file **CCountry.cpp**.

◆ operator>>() [2/2]

```
std::istream& operator>> ( std::istream &                     is,
                           Inheritor_��ountry_second &  Inheritor_�ountry_second
                         )
```

Definition at line **318** of file **CCountry.cpp**.

CCountry.h File Reference

```
#include <iostream>
#include <string>
#include <fstream>
#include <sstream>
#include <regex>
#include <iomanip>
```

Go to the source code of this file.

Classes

class	**C�itizen**
class	**CCountry**
class	**Inheritor_��ountry**
class	**Inheritor_��ountry_second**

Typedefs

typedef int	**Cint**

Functions

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_��ountry** &Inheritor_�ountry1, const **Inheritor_��ountry** &Inheritor_�ountry2)
bool	**operator!=** (const **Inheritor_��ountry** &Inheritor_�ountry1, const **Inheritor_��ountry** &Inheritor_�ountry2)
bool	**operator==** (const **Inheritor_��ountry_second** &Inheritor_�ountry_second1, const **Inheritor_��ountry_second** &Inheritor_�ountry_second2)
bool	**operator!=** (const **Inheritor_��ountry_second** &Inheritor_�ountry_second1, const **Inheritor_��ountry_second** &Inheritor_�ountry_second2)
std::ostream &	**operator<<** (std::ostream &os, const **Inheritor_��ountry** &Inheritor_�ountry)
std::ostream &	**operator<<** (std::ostream &os, const **Inheritor_��ountry_second** &Inheritor_�ountry_second)
std::istream &	**operator>>** (std::istream &is, **Inheritor_��ountry** &Inheritor_�ountry)
std::istream &	**operator>>** (std::istream &is, **Inheritor_��ountry_second** &Inheritor_�ountry_second)
std::ostream &	**operator<<** (std::ostream &os, const **CCountry** &Country)

Typedef Documentation

◆ Cint

typedef int **Cint**

Definition at line **8** of file **CCountry.h**.

Function Documentation

◆ check_str()

```
bool check_str ( const std::string &  str )
```

Definition at line **198** of file **CCountry.cpp**.

◆ operator!=() [1/3]

```
bool operator!= ( const CCountry &  Country1,
                  const CCountry &  Country2
                )
```

Definition at line **116** of file **CCountry.cpp**.

◆ operator!=() [2/3]

```
bool operator!= ( const Inheritor_��ountry &  Inheritor_�ountry1,
                  const Inheritor_��ountry &  Inheritor_�ountry2
                )
```

Definition at line **155** of file **CCountry.cpp**.

◆ operator!=() [3/3]

```
bool operator!= ( const Inheritor_��ountry_second &  Inheritor_�ountry_second1,
                  const Inheritor_��ountry_second &  Inheritor_�ountry_second2
                )
```

Definition at line **194** of file **CCountry.cpp**.

◆ operator<<() [1/3]

```
std::ostream& operator<< ( std::ostream &       os,
                           const CCountry &     Country
                         )
```

Definition at line **222** of file **CCountry.cpp**.

◆ operator<<() [2/3]

```
std::ostream& operator<< ( std::ostream &                os,
                           const Inheritor_��ountry &    Inheritor_�ountry
                         )
```

Definition at line **226** of file **CCountry.cpp**.

◆ operator<<() [3/3]

```
std::ostream& operator<< ( std::ostream &                    os,
                           const Inheritor_��ountry_second &  Inheritor_�ountry_second
                         )
```

Definition at line **230** of file **CCountry.cpp**.

◆ operator==() [1/3]

```
bool operator== ( const CCountry &  Country1,
                  const CCountry &  Country2
                )
```

Definition at line **85** of file **CCountry.cpp**.

◆ operator==() [2/3]

```
bool operator== ( const Inheritor_��ountry &  Inheritor_�ountry1,
                  const Inheritor_��ountry &  Inheritor_�ountry2
                )
```

Definition at line **120** of file **CCountry.cpp**.

◆ operator==() [3/3]

```
bool operator== ( const Inheritor_��ountry_second &  Inheritor_�ountry_second1,
                  const Inheritor_��ountry_second &  Inheritor_�ountry_second2
                )
```

Definition at line **159** of file **CCountry.cpp**.

◆ operator>>() [1/2]

```
std::istream& operator>> ( std::istream &          is,
                           Inheritor_��ountry &    Inheritor_�ountry
                         )
```

Definition at line **234** of file **CCountry.cpp**.

◆ operator>>() [2/2]

```
std::istream& operator>> ( std::istream &                  is,
                           Inheritor_��ountry_second &     Inheritor_�ountry_second
                         )
```

Definition at line **318** of file **CCountry.cpp**.

CMetod.cpp File Reference

```
#include "CMetod.h"
```

Go to the source code of this file.

Functions

bool	**sortTitle** (**CCountry** &a, **CCountry** &b)
bool	**sortNumber_of_cities** (**CCountry** &a, **CCountry** &b)
bool	**sortPopulation** (**CCountry** &a, **CCountry** &b)
bool	**sortArea** (**CCountry** &a, **CCountry** &b)
bool	**sortCitizen** (**CCountry** &a, **CCountry** &b)
std::istream &	**operator>>** (std::istream &is, **CMetod** &Cmetod)
std::ostream &	**operator<<** (std::ostream &os, **CMetod** &Cmetod)

Function Documentation

◆ operator<<()

```
std::ostream& operator<< ( std::ostream &  os,
                           CMetod &        Cmetod
                         )
```

Definition at line **340** of file **CMetod.cpp**.

◆ sortCitizen()

```
bool sortCitizen ( CCountry &  a,
                   CCountry &  b
                 )
```

Definition at line **316** of file **CMetod.cpp**.

◆ operator>>()

```
std::istream& operator>> ( std::istream &  is,
                           CMetod &        Cmetod
                         )
```

Definition at line **320** of file **CMetod.cpp**.

◆ sortNumber_of_cities()

```
bool sortNumber_of_cities ( CCountry &  a,
                            CCountry &  b
                          )
```

Definition at line **304** of file **CMetod.cpp**.

◆ sortArea()

```
bool sortArea ( CCountry &  a,
                CCountry &  b
              )
```

Definition at line **312** of file **CMetod.cpp**.

◆ sortPopulation()

```
bool sortPopulation ( CCountry &  a,
                      CCountry &  b
                    )
```

Definition at line **308** of file **CMetod.cpp**.

◆ sortTitle()

```
bool sortTitle ( CCountry &  a,
                 CCountry &  b
               )
```

Definition at line **300** of file **CMetod.cpp**.

CMetod.h File Reference

```
#include "CCountry.h"
```

Go to the source code of this file.

Classes

class	**CMetod**

Functions

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, **CMetod** &Cmetod)
std::ostream &	**operator<<** (std::ostream &os, **CMetod** &Cmetod)

Function Documentation

◆ operator<<()

std::ostream& operator<< (std::ostream & os,
CMetod & Cmetod
)

Definition at line **340** of file **CMetod.cpp**.

◆ operator>>()

std::istream& operator>> (std::istream & is,
CMetod & Cmetod
)

Definition at line **320** of file **CMetod.cpp**.

◆ sortArea()

bool sortArea (**CCountry** & ,
CCountry &
)

Definition at line **312** of file **CMetod.cpp**.

◆ sortCitizen()

bool sortCitizen (**CCountry** & ,
CCountry &
)

Definition at line **316** of file **CMetod.cpp**.

◆ sortNumber_of_cities()

bool sortNumber_of_cities (**CCountry** & ,
CCountry &
)

Definition at line **304** of file **CMetod.cpp**.

◆ sortPopulation()

bool sortPopulation (**CCountry** & ,
CCountry &
)

Definition at line **308** of file **CMetod.cpp**.

menu.cpp File Reference

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

Go to the source code of this file.

Functions

void **menu** ()

Function Documentation

◆ sortTitle()

bool sortTitle (**CCountry** & ,
CCountry &
)

Definition at line **300** of file **CMetod.cpp**.

◆ menu()

void menu ( )

[illegible].

Definition at line **2** of file **menu.cpp**.

menu.h File Reference

`#include "CMetod.h"`

Go to the source code of this file.

Functions

void **menu** ()

Function Documentation

menu()

void menu ( )

**********.

Definition at line **2** of file **menu.cpp**.

tests.cpp File Reference

`#include "menu.h"`

Go to the source code of this file.

Macros

#define **_CRTDBG_MAP_ALLOC**

Functions

int **main** ()

Macro Definition Documentation

_CRTDBG_MAP_ALLOC

#define _CRTDBG_MAP_ALLOC

Definition at line **3** of file **tests.cpp**.

РГЗ.cpp File Reference

```
#include <iostream>
#include "menu.h"
```

Go to the source code of this file.

Macros

#define _CRTDBG_MAP_ALLOC

Functions

int main ()

Function Documentation

◆ main()

int main ( )

Definition at line 5 of file tests.cpp.

Macro Definition Documentation

◆ _CRTDBG_MAP_ALLOC

#define _CRTDBG_MAP_ALLOC

Definition at line 3 of file РГЗ.cpp.

Function Documentation

◆ main()

int main ( )

[illegible].

[illegible].

Definition at line 5 of file РГЗ.cpp.

Код програми:

Тест.cpp

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

int main()
{
    menu(); /// Функция меню.
    if (_CrtDumpMemoryLeaks()) /// Проверка на утечку памяти.
    {
        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 <iomanip>
typedef int Cint;
class CCitizen
{
private:
    bool place_of_birth_citizen;
    Cint birthday_citizen;
public:
    const bool getPlace_of_birth_citizen() const;
    Cint getBirthday_citizen() const;
    void setPlace_of_birth_citizen(const bool&);
    void setBirthday_citizen(const int&);
};
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
int&);
    ~CCountry();
    virtual bool getPlace_of_birth_citizen() const;
    virtual Cint 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 int&);
    virtual bool getMonarchy() const;
    virtual std::string getInfo() const = 0;
    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());
    }
    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 override 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 int&, const bool&);
    ~Inheritor_CCountry();
    virtual std::string getInfo() const final;
    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;
    }
    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 int&, const bool&);
```

```
    ~Inheritor_CCountry_second();
    virtual std::string getInfo() const final;
    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;
    }
    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);
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(); }
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::setPlace_of_birth_citizen(const bool& 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(0);
	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 bool& Place_of_birth_citizen, const int&
Birthday_citizen)
{
	title = Title;
	number_of_cities = Number_of_cities;
	population = Population;
	area = Area;
	citizen.setPlace_of_birth_citizen(Place_of_birth_citizen);
	citizen.setBirthday_citizen(Birthday_citizen);
	population_density = Area / Population;
	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; }
Cint 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 int& Birthday_citizen) { birthday_citizen =
Birthday_citizen; }
bool CCountry::getMonarchy() const { return false; }
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(false) { 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 int& Birthday_citizen, const bool& Monarchy) : CCountry(Title,
Number_of_cities, Population, Area, Place_of_birth_citizen, Birthday_citizen),
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(false) { 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 int& Birthday_citizen, const bool& Gross_domestic_product) :
CCountry(Title, Number_of_cities, Population, Area, Place_of_birth_citizen,
Birthday_citizen), 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.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.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.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.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.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;
}
```

CMetod.h

```
#pragma once
#include "CCountry.h"
class CMetod
{
    /// Класс - массив.
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();
    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&));
    CCountry* operator[](const int& index)
    {
        return countries[index];
    }
    friend std::ostream& operator<< (std::ostream& os, CMetod& Cmetod);
};
```

```
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, CMetod& Cmetod);
std::ostream& operator<< (std::ostream& os, CMetod& Cmetod);
dir.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++;
	}
	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++;
	}
}
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++;
	}
	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++;
```

```
	}
}
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;
	}

}
void CMetod::get_to_Screen(const int& index) const
{
	std::cout << "Title " << "Number_of_cities " << "Population " << "Area " <<
"Place_of_birth_citizen " << "Birthday_citizen " << "GetInfo" << "Kol_nationality" << "\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() << " " << countries[index]->getInfo();
	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;
		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;
		int Birthday_citizen;
```

```
			s >> Birthday_citizen;
			int i7 = str.find(" ", i6 + 1);
			s.clear();
			temp = str.substr(i6 + 1, i7 - i6);
			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, 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, 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-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()));
}
std::istream& operator>> (std::istream& is, CMetod& Cmetod) {
	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;
}
```

menu.h

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


void menu();
```

menu.cpp

```
#include "menu.h"
void menu()
{
    setlocale(LC_ALL, "Russian"); /// Локализация консоли.
    int n = 0, temp_i;
    int m;
    CMetod dir;
    std::ifstream f("text.txt");
    std::ofstream d;
    Inheritor_CCountry_second firstcountry1("Страна1", 143, 45745656, 47342362, 1, 22062012,
0);
    dir.add_el(firstcountry1);
    Inheritor_CCountry_second firstcountry2("Страна2", 156, 38567454, 68457458, 1, 13012016,
1);
    dir.add_el(firstcountry2);
    Inheritor_CCountry_second firstcountry3("Страна3", 167, 46357625, 98686453, 1, 31102007,
0);
    dir.add_el(firstcountry3);
    Inheritor_CCountry_second firstcountry4("Страна4", 179, 78567583, 68457458, 1, 27072000,
0);
    dir.add_el(firstcountry4);
    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)
        {
                        Inheritor_CCountry_second firstcountry5("Страна5", 323, 93645665,
78767464, 1, 24112001, 1);
            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)
        {
            std::cout << dir;
        }
        else if (n == 4)
        {
            dir.find_to_population_density();
        }
        else if (n == 5)
        {
            d.open("text.txt");
            d << dir;
            d.close();
            std::cout << "Данные записаны в файл." << "\n";
        }
        else if (n == 6)
        {
            f.open("text.txt");
            f >> dir;
            f.close();
            std::cout << "Данные считаны из файла." << "\n";
        }
        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();
}
```

tests.cpp

```
#pragma once
#include "menu.h"
#define _CRTDBG_MAP_ALLOC

int main()
{
	setlocale(LC_ALL, "Russian");
	std::ifstream in("test1.txt");
	std::ofstream on("test2.txt");
	int n;
	CMetod dir1;
	in >> dir1;
	on << dir1;
	std::cout << "Если содержимое файлов test1 и test2 теперь совпадают, то тест пройден.
Нажмите любую клавишу для завершения работы программы. ";
	if (_CrtDumpMemoryLeaks())
	{
		std::cout << "Утечка памяти обнаружена." << "\n";
	}
	else
	{
		std::cout << "Утечка памяти не обнаружена." << "\n";
	}
	std::cin >> n;
}
```

text.txt

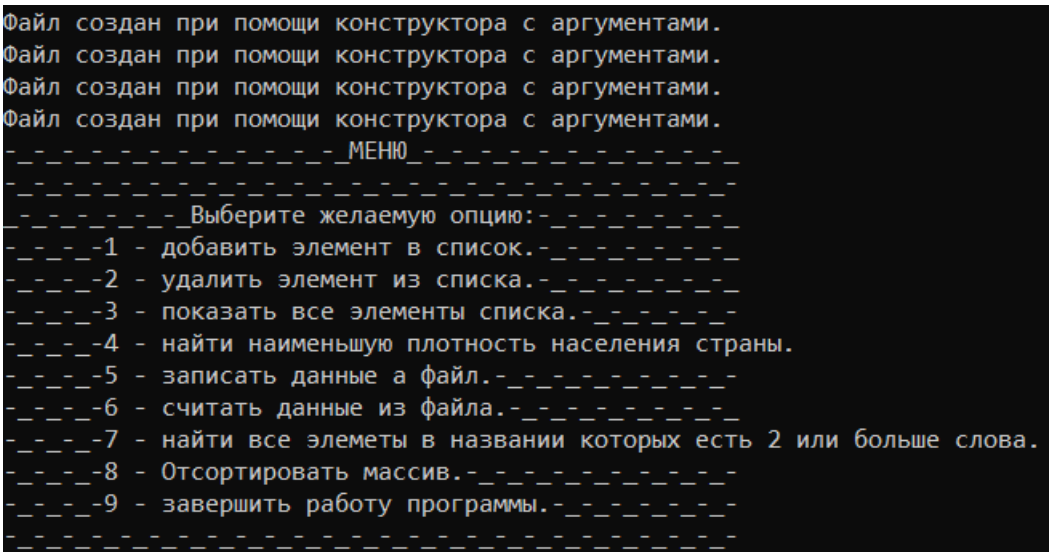
```
1 _Страна1_ 143 45745656 47342362 1 22062012 0
2 _Страна2_ 156 38567454 68457458 0 13012016 1
1 _Страна3_ 167 46357625 98686453 1 31102007 0
2 _Страна4_ 179 78567583 68457458 0 27072000 0
1 _Страна1_ 143 45745656 47342362 1 22062012 0
2 _Страна2_ 156 38567454 68457458 0 13012016 1
1 _Страна3_ 167 46357625 98686453 1 31102007 0
2 _Страна4_ 179 78567583 68457458 0 27072000 0
```

tests2.txt

```
1 _Страна1_ 143 45745656 47342362 1 22062012 0
2 _Страна2_ 156 38567454 68457458 0 13012016 1
1 _Страна3_ 167 46357625 98686453 1 31102007 0
2 _Страна4_ 179 78567583 68457458 0 27072000 0
```

Додаток Б

Результати роботи програми



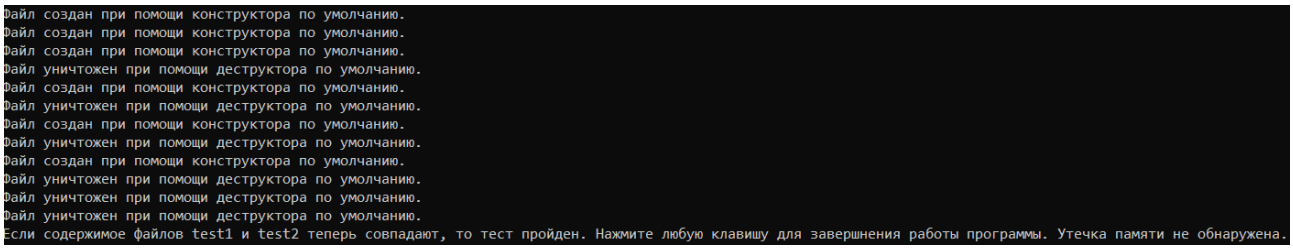


Рисунок 1 – Результати роботи програми