ГУАП

КАФЕДРА № 14

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ассистент |  |  |  | А. Ю. Петров |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №1 |
|  |
| по курсу: ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 1142 |  |  |  | А.Н. Коновалов |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2023

1. **Постановка задачи**

Разработать класс Keeper, который реализует контейнер для хранения и обработки объектов. Класс Keeper должен обеспечивать следующие функции:

- добавление и удаление производных объектов абстрактного класса Base (базовый класс определяется согласно варианту задания);

- полное сохранение себя в файле;

- полное восстановление себя из файла.

Класс Фамильное дерево хранит данные о семье. Каждый член семьи имеет:

ФИО, знания о родителях, знания о супруге, знания о своих детях, дату рождения, дату смерти (если есть), возраст.

1. **Формализация задачи**

Данная программа разбита на 14 файлов: 7 заголовочных файлов (Children.h, GrandParents.h, Keeper.h, Menu.h, Parents.h, Person.h, You.h); 6 реализационных файлов (Children.cpp, GrandParents.cpp, Keeper.cpp, Menu.cpp, Parents.cpp, You.cpp) и один главный (Main.cpp) – это файлы с реализацией контейнера для хранения и обработки объектов, объявлением, определением методов класса соответственно, а также файл с управляющей функцией.

**Файл Children.h состоит из:**

- Определения геттеров и сеттеров для полей класса

- Конструктора по умолчанию, с параметрами и копирования

- Определения виртуальных методов класса Person

**Файл GrandParents.h состоит из:**

- Определения геттеров и сеттеров для полей класса

- Конструктора по умолчанию, с параметрами и копирования

- Определения виртуальных методов класса Person

**Файл Keeper.h состоит из:**

- Поле size, которое содержит данные элемента и поле head, которое хранит адрес головы.

- Конструктора по умолчанию (без параметров), конструктора с параметрами, деструктора.

- Определения методов, реализованных в файле Keeper.cpp

**Файл Menu.h состоит из:**

Содержит определение методов: main\_menu – запускает меню , show\_options – выводит в консоль пункты меню, show\_all\_persons – выводит все наши обьекты в консоль, change\_data – изменение данных у выбранного обьекта, save\_to\_file – сохранение в файл, load\_from\_file – выгрузка данных из файла , delete\_person –удалить объект, exit\_from\_program – выход из программы

**Файл Parents.h состоит из:**

- Определения геттеров и сеттеров для полей класса

- Конструктора по умолчанию, с параметрами и копирования

- Определения виртуальных методов класса Person

**Файл Person.h состоит из:**

Определения абстрактного базового класса Person Определение виртуальных методов

- show – для вывода данных в консоль

- rewrite – для изменения данных

- saving – для сохранения данных

- redact\_str – для редактирования полей

**Файл You.h состоит из:**

- Определения геттеров и сеттеров для полей класса

- Конструктора по умолчанию, с параметрами и копирования

- Определения виртуальных методов класса Person

**Файл Children.cpp состоит из:**

- Конструктора по умолчанию, конструктора с параметрами, конструктора копирования

- Реализации геттеров и сеттеров для полей класса

- Реализацию методов из абстрактного класса Person

**Файл GrandParents.cpp состоит из:**

- Конструктора по умолчанию, конструктора с параметрами, конструктора копирования

- Реализации геттеров и сеттеров для полей класса

- Реализацию методов из абстрактного класса Person

**Файл Keeper.cpp состоит из:**

- Метод insert – добавление нового элемента в массив объектов, динамическое выделение памяти.

- Метод remove – удаляется выбранный объект из контейнера, освобождается память и уменьшается размер массива объектов, в случае если выбранного элемента нет, то выбрасывается оповещение о некорректном вводе.

- Метод save – сохранение в текстовый файл всех объектов, которые пользователь ввел с клавиатуры.

- Метод load – чтение из файла и заполнение контейнера данными.

- Перегрузка оператора [] – для прохода по связному списку.

- Метод get\_size – получение количества элементов в контейнере.

**Файл Menu.cpp состоит из:**

- Реализацию методов определенных в Menu.h

**Файл Parents.cpp состоит из:**

- Конструктора по умолчанию, конструктора с параметрами, конструктора копирования

- Реализации геттеров и сеттеров для полей класса

- Реализацию методов из абстрактного класса Person

**Файл You.cpp состоит из:**

- Конструктора по умолчанию, конструктора с параметрами, конструктора копирования

- Реализации геттеров и сеттеров для полей класса

- Реализацию методов из абстрактного класса Person

В программе возможно добавление, удаление и редактирование членов семьи четырёх категорий: Вас, ваших детей, родителей, и родителей родителей. Каждая категория имеет свои соответствующие характеристики, которые заполняются пользователем: полное имя, родители, супруг/супруга, дети, дата рождения, смертии(при наличии) и возраст. Все вносимые пользователем данные можно загрузить в файл data.txt, а также выгрузить из него. Есть возможность продемонстрировать всех созданных людей. В программе реализовано пользовательское меню выбора действий для работы:

1 - Показать всех членов семьи

2 - Изменить данные о человеке

3 - Добавить нового члена семьи

4 - Сохранить все данные в файле

5 - Загрузить из файла данные о семье

6 - Удалить члена семьи

0 - Выход из программы

1. **Таблица описания классов**

Таблица.1 – Описание классов

|  |  |  |  |
| --- | --- | --- | --- |
| Название класса | Назначение класса | Наследование | Тип |
| Keeper | Класс- контейнер содержащий производные объекты класса Person | - | - |
| Person | Абстрактный базовый  класс для классов,  хранящихся в Keeper | - | - |
| Parents | Содержимое Keeper, хранит данные о родителях | Person | public |
| Grandparents | Содержимое Keeper, хранит данные о родителях родителей | Person | public |
| Children | Содержимое Keeper, хранит данные о детях | Person | public |
| You | Содержимое Keeper, хранит данные о вас | Person | public |

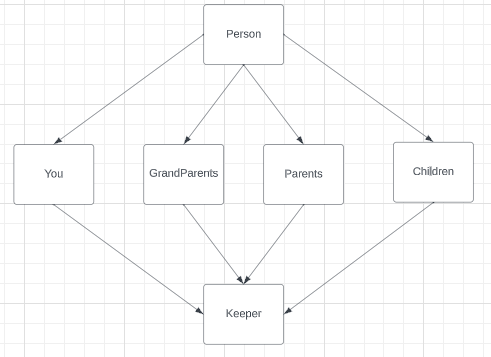


Рисунок 1- Схема иерархии классов

1. **Исходный код**

You.cpp

#include <iostream>

#include <fstream>

#include <string>

#include "You.h"

using namespace std;

You::You() { cout << "Constructor You without attributes was called." << endl; }

You::~You() { cout << "Destructor You was called." << endl; }

You::You(const You& You\_copy) {

fullName = You\_copy.fullName;

parents = You\_copy.parents;

spouses = You\_copy.spouses;

children = You\_copy.children;

dateBirh = You\_copy.dateBirh;

dateDeath = You\_copy.dateDeath;

age = You\_copy.age;

cout << "Constructor copy You was called." << endl;

}

You::You(string fullName)

{

this->fullName = fullName; cout << "Constructor You with attributes was called." << endl;

}

string You::get\_fullName() const

{

return fullName;

}

void You::set\_fullName(string name\_ch)

{

fullName = name\_ch;

}

string You::get\_parents() const

{

return parents;

}

void You::set\_parents(string post\_ch)

{

parents = post\_ch;

}

string You::get\_spouses() const

{

return spouses;

}

void You::set\_spouses(string number\_ch)

{

spouses = number\_ch;

}

string You::get\_children() const

{

return children;

}

void You::set\_children(string children\_ch)

{

children = children\_ch;

}

string You::get\_dateBirh() const

{

return dateBirh;

}

void You::set\_dateBirh(string dateBirh\_ch)

{

dateBirh = dateBirh\_ch;

}

string You::get\_dateDeath() const

{

return dateDeath;

}

void You::set\_dateDeath(string dateDeath\_ch)

{

dateDeath = dateDeath\_ch;

}

string You::get\_age() const

{

return age;

}

void You::set\_age(string age\_ch)

{

age = age\_ch;

}

void You::show() {

cout << "------------------------------------" << endl;

cout << "Person: You" << endl;

cout << "Full Name: " << fullName << endl;

cout << "Parents: " << parents << endl;

cout << "Spouses: " << spouses << endl;

cout << "You " << children << endl;

cout << "Date Birhday " << dateBirh << endl;

cout << "Date Death " << dateDeath << endl;

cout << "Age " << age << endl;

cout << "------------------------------------" << endl;

}

void You::rewrite()

{

cout << "------------------------------------" << endl;

cout << "Person: You" << endl;

cout << "Input new fullName: ";

getline(cin, fullName);

getline(cin, fullName);

cout << "Input parents: ";

getline(cin, parents);

cout << "Input spouses: ";

getline(cin, spouses);

cout << "Input children: ";

getline(cin, children);

cout << "Input date birhday: ";

getline(cin, dateBirh);

cout << "Input date death: ";

getline(cin, dateDeath);

cout << "Input age: ";

getline(cin, age);

cout << "------------------------------------" << endl;

}

void You::saving() {

ofstream outfile;

string initfile = "data.txt";

outfile.open(initfile, ios\_base::app);

if (!outfile.is\_open())

{

cout << "Error opening!" << endl;

exit(1);

}

outfile << 4 << endl << fullName << endl << parents << endl << spouses << endl << children << endl << dateBirh << endl << dateDeath << endl << age << endl;

outfile.close();

}

void You::redact\_str(int num\_str, string red\_str) {

switch (num\_str)

{

case 1:

this->fullName = red\_str;

break;

case 2:

this->parents = red\_str;

break;

case 3:

this->spouses = red\_str;

break;

case 4:

this->children = red\_str;

break;

case 5:

this->dateBirh = red\_str;

break;

case 6:

this->dateDeath = red\_str;

break;

case 7:

this->age = red\_str;

break;

default:

throw "Incorrect number of string!";

break;

}

}

You.h

#pragma once

#include "Person.h"

using namespace std;

class You : public Person {

private:

string fullName;

string parents;

string spouses;

string children;

string dateBirh;

string dateDeath;

string age;

public:

You();

~You();

You(const You& You\_copy);

You(string fullName);

string get\_fullName() const;

void set\_fullName(string fullName\_ch);

string get\_parents() const;

void set\_parents(string parents\_ch);

string get\_spouses() const;

void set\_spouses(string spouses\_ch);

string get\_children() const;

void set\_children(string children\_ch);

string get\_dateBirh() const;

void set\_dateBirh(string dateBirh\_ch);

string get\_dateDeath() const;

void set\_dateDeath(string dateDeath\_ch);

string get\_age() const;

void set\_age(string age\_ch);

virtual void show();

virtual void rewrite();

virtual void saving();

virtual void redact\_str(int num\_str, string red\_str);

};

Children.cpp

#include <iostream>

#include <fstream>

#include <string>

#include "Children.h"

using namespace std;

Children::Children() { cout << "Constructor Children without attributes was called." << endl; }

Children::~Children() { cout << "Destructor Children was called." << endl; }

Children::Children(const Children& Children\_copy) {

fullName = Children\_copy.fullName;

parents = Children\_copy.parents;

spouses = Children\_copy.spouses;

children = Children\_copy.children;

dateBirh = Children\_copy.dateBirh;

dateDeath = Children\_copy.dateDeath;

age = Children\_copy.age;

cout << "Constructor copy Children was called." << endl;

}

Children::Children(string fullName)

{

this->fullName = fullName; cout << "Constructor Children with attributes was called." << endl;

}

string Children::get\_fullName() const

{

return fullName;

}

void Children::set\_fullName(string name\_ch)

{

fullName = name\_ch;

}

string Children::get\_parents() const

{

return parents;

}

void Children::set\_parents(string post\_ch)

{

parents = post\_ch;

}

string Children::get\_spouses() const

{

return spouses;

}

void Children::set\_spouses(string number\_ch)

{

spouses = number\_ch;

}

string Children::get\_children() const

{

return children;

}

void Children::set\_children(string children\_ch)

{

children = children\_ch;

}

string Children::get\_dateBirh() const

{

return dateBirh;

}

void Children::set\_dateBirh(string dateBirh\_ch)

{

dateBirh = dateBirh\_ch;

}

string Children::get\_dateDeath() const

{

return dateDeath;

}

void Children::set\_dateDeath(string dateDeath\_ch)

{

dateDeath = dateDeath\_ch;

}

string Children::get\_age() const

{

return age;

}

void Children::set\_age(string age\_ch)

{

age = age\_ch;

}

void Children::show() {

cout << "------------------------------------" << endl;

cout << "Person: Children" << endl;

cout << "Full Name: " << fullName << endl;

cout << "Parents: " << parents << endl;

cout << "Spouses: " << spouses << endl;

cout << "Children " << children << endl;

cout << "Date Birhday " << dateBirh << endl;

cout << "Date Death " << dateDeath << endl;

cout << "Age " << age << endl;

cout << "------------------------------------" << endl;

}

void Children::rewrite()

{

cout << "------------------------------------" << endl;

cout << "Person: Children" << endl;

cout << "Input new fullName: ";

getline(cin, fullName);

getline(cin, fullName);

cout << "Input parents: ";

getline(cin, parents);

cout << "Input spouses: ";

getline(cin, spouses);

cout << "Input children: ";

getline(cin, children);

cout << "Input date birhday: ";

getline(cin, dateBirh);

cout << "Input date death: ";

getline(cin, dateDeath);

cout << "Input age: ";

getline(cin, age);

cout << "------------------------------------" << endl;

}

void Children::saving() {

ofstream outfile;

string initfile = "data.txt";

outfile.open(initfile, ios\_base::app);

if (!outfile.is\_open())

{

cout << "Error opening!" << endl;

exit(1);

}

outfile << 1 << endl << fullName << endl << parents << endl << spouses << endl << children << endl << dateBirh << endl << dateDeath << endl << age << endl;

outfile.close();

}

void Children::redact\_str(int num\_str, string red\_str) {

switch (num\_str)

{

case 1:

this->fullName = red\_str;

break;

case 2:

this->parents = red\_str;

break;

case 3:

this->spouses = red\_str;

break;

case 4:

this->children = red\_str;

break;

case 5:

this->dateBirh = red\_str;

break;

case 6:

this->dateDeath = red\_str;

break;

case 7:

this->age = red\_str;

break;

default:

throw "Incorrect number of string!";

break;

}

}

Children,h

#pragma once

#include "Person.h"

using namespace std;

class Children : public Person {

private:

string fullName;

string parents;

string spouses;

string children;

string dateBirh;

string dateDeath;

string age;

public:

Children();

~Children();

Children(const Children& Children\_copy);

Children(string fullName);

string get\_fullName() const;

void set\_fullName(string fullName\_ch);

string get\_parents() const;

void set\_parents(string parents\_ch);

string get\_spouses() const;

void set\_spouses(string spouses\_ch);

string get\_children() const;

void set\_children(string children\_ch);

string get\_dateBirh() const;

void set\_dateBirh(string dateBirh\_ch);

string get\_dateDeath() const;

void set\_dateDeath(string dateDeath\_ch);

string get\_age() const;

void set\_age(string age\_ch);

virtual void show();

virtual void rewrite();

virtual void saving();

virtual void redact\_str(int num\_str, string red\_str);

};

Grandparents.cpp

#include <iostream>

#include <fstream>

#include <string>

#include "GrandParents.h"

using namespace std;

GrandParents::GrandParents()

{

cout << "Constructor GrandParents without attributes was called." << endl;

}

GrandParents::~GrandParents()

{

cout << "Destructor GrandParents was called." << endl;

}

GrandParents::GrandParents(const GrandParents& GrandParents\_copy)

{

fullName = GrandParents\_copy.fullName;

parents = GrandParents\_copy.parents;

spouses = GrandParents\_copy.spouses;

children = GrandParents\_copy.children;

dateBirh = GrandParents\_copy.dateBirh;

dateDeath = GrandParents\_copy.dateDeath;

age = GrandParents\_copy.age;

cout << "Constructor copy was called." << endl;

}

GrandParents::GrandParents(string fullName)

{

this->fullName = fullName; cout << "Constructor GrandParents with attributes was called." << endl;

}

string GrandParents::get\_fullName() const

{

return fullName;

}

void GrandParents::set\_fullName(string name\_ch)

{

fullName = name\_ch;

}

string GrandParents::get\_parents() const

{

return parents;

}

void GrandParents::set\_parents(string post\_ch)

{

parents = post\_ch;

}

string GrandParents::get\_spouses() const

{

return spouses;

}

void GrandParents::set\_spouses(string number\_ch)

{

spouses = number\_ch;

}

string GrandParents::get\_children() const

{

return children;

}

void GrandParents::set\_children(string children\_ch)

{

children = children\_ch;

}

string GrandParents::get\_dateBirh() const

{

return dateBirh;

}

void GrandParents::set\_dateBirh(string dateBirh\_ch)

{

dateBirh = dateBirh\_ch;

}

string GrandParents::get\_dateDeath() const

{

return dateDeath;

}

void GrandParents::set\_dateDeath(string dateDeath\_ch)

{

dateDeath = dateDeath\_ch;

}

string GrandParents::get\_age() const

{

return age;

}

void GrandParents::set\_age(string age\_ch)

{

age = age\_ch;

}

void GrandParents::show() {

cout << "------------------------------------" << endl;

cout << "Person: GrandParents" << endl;

cout << "Full Name: " << fullName << endl;

cout << "Parents: " << parents << endl;

cout << "Spouses: " << spouses << endl;

cout << "Children " << children << endl;

cout << "Date Birhday " << dateBirh << endl;

cout << "Date Death " << dateDeath << endl;

cout << "Age " << age << endl;

cout << "------------------------------------" << endl;

}

void GrandParents::rewrite()

{

cout << "------------------------------------" << endl;

cout << "Person: GrandParents" << endl;

cout << "Input fullName: ";

getline(cin, fullName);

getline(cin, fullName);

cout << "Input parents: ";

getline(cin, parents);

cout << "Input spouses: ";

getline(cin, spouses);

cout << "Input children: ";

getline(cin, children);

cout << "Input date birhday: ";

getline(cin, dateBirh);

cout << "Input date death: ";

getline(cin, dateDeath);

cout << "Input age: ";

getline(cin, age);

cout << "------------------------------------" << endl;

}

void GrandParents::saving() {

ofstream outfile;

string initfile = "data.txt";

outfile.open(initfile, ios\_base::app);

if (!outfile.is\_open())

{

cout << "Error opening the outfile << endl!";

exit(1);

}

outfile << 3 << endl << fullName << endl << parents << endl << spouses << endl << children << endl << dateBirh << endl << dateDeath << endl << age << endl;

outfile.close();

}

void GrandParents::redact\_str(int num\_str, string red\_str) {

switch (num\_str)

{

case 1:

this->fullName = red\_str;

break;

case 2:

this->parents = red\_str;

break;

case 3:

this->spouses = red\_str;

break;

case 4:

this->children = red\_str;

break;

case 5:

this->dateBirh = red\_str;

break;

case 6:

this->dateDeath = red\_str;

break;

case 7:

this->age = red\_str;

break;

default:

cout << "Incorrect number of string!" << endl;

break;

}

}

Grandparents.h

#pragma once

#include "Person.h"

using namespace std;

class GrandParents : public Person {

private:

string fullName;

string parents;

string spouses;

string children;

string dateBirh;

string dateDeath;

string age;

public:

GrandParents();

~GrandParents();

GrandParents(const GrandParents& GrandParents\_copy);

GrandParents(string fullName);

string get\_fullName() const;

void set\_fullName(string fullName\_ch);

string get\_parents() const;

void set\_parents(string parents\_ch);

string get\_spouses() const;

void set\_spouses(string spouses\_ch);

string get\_children() const;

void set\_children(string children\_ch);

string get\_dateBirh() const;

void set\_dateBirh(string dateBirh\_ch);

string get\_dateDeath() const;

void set\_dateDeath(string dateDeath\_ch);

string get\_age() const;

void set\_age(string age\_ch);

virtual void show();

virtual void rewrite();

virtual void saving();

virtual void redact\_str(int num\_str, string red\_str);

};

Parents.cpp

//#include "libra.h"

#include <iostream>

#include <fstream>

#include <string>

#include "Parents.h"

using namespace std;

Parents::Parents()

{

cout << "Constructor Parents without attributes was called." << endl;

}

Parents::~Parents() { cout << "Destructor Parents was called." << endl; }

Parents::Parents(const Parents& Parents\_copy) {

fullName = Parents\_copy.fullName;

parents = Parents\_copy.parents;

spouses = Parents\_copy.spouses;

children = Parents\_copy.children;

dateBirh = Parents\_copy.dateBirh;

dateDeath = Parents\_copy.dateDeath;

age = Parents\_copy.age;

cout << "Constructor copy Parents was called." << endl;

}

Parents::Parents(string fullName)

{

this->fullName = fullName; cout << "Constructor Parents with attributes was called." <<endl;

}

string Parents::get\_fullName() const

{

return fullName;

}

void Parents::set\_fullName(string name\_ch)

{

fullName = name\_ch;

}

string Parents::get\_parents() const

{

return parents;

}

void Parents::set\_parents(string post\_ch)

{

parents = post\_ch;

}

string Parents::get\_spouses() const

{

return spouses;

}

void Parents::set\_spouses(string number\_ch)

{

spouses = number\_ch;

}

string Parents::get\_children() const

{

return children;

}

void Parents::set\_children(string children\_ch)

{

children = children\_ch;

}

string Parents::get\_dateBirh() const

{

return dateBirh;

}

void Parents::set\_dateBirh(string dateBirh\_ch)

{

dateBirh = dateBirh\_ch;

}

string Parents::get\_dateDeath() const

{

return dateDeath;

}

void Parents::set\_dateDeath(string dateDeath\_ch)

{

dateDeath = dateDeath\_ch;

}

string Parents::get\_age() const

{

return age;

}

void Parents::set\_age(string age\_ch)

{

age = age\_ch;

}

void Parents::show() {

cout << "------------------------------------" << endl;

cout << "Person: Parents" << endl;

cout << "Full Name: " << fullName << endl;

cout << "Parents: " << parents << endl;

cout << "Spouses: " << spouses << endl;

cout << "Children " << children << endl;

cout << "Date Birhday " << dateBirh << endl;

cout << "Date Death " << dateDeath << endl;

cout << "Age " << age << endl;

cout << "------------------------------------" << endl;

}

void Parents::rewrite() {

cout << "------------------------------------" << endl;

cout << "Parents: " << endl;

cout << "Input new fullName: ";

getline(cin, fullName);

getline(cin, fullName);

cout << "Input parents: ";

getline(cin, parents);

cout << "Input spouses: ";

getline(cin, spouses);

cout << "Input children: ";

getline(cin, children);

cout << "Input date birhday: ";

getline(cin, dateBirh);

cout << "Input date death: ";

getline(cin, dateDeath);

cout << "Input age: ";

getline(cin, age);

cout << "------------------------------------" << endl;

}

void Parents::saving() {

ofstream outfile;

string initfile = "data.txt";

outfile.open(initfile, ios\_base::app);

if (!outfile.is\_open())

{

cout << "Error opening the outfile!" << endl;

exit(1);

}

else

{

outfile << 2 << endl << fullName << endl << parents << endl << spouses << endl << children << endl << dateBirh << endl << dateDeath << endl << age << endl;

outfile.close();

}

}

void Parents::redact\_str(int num\_str, string red\_str) {

switch (num\_str)

{

case 1:

this->fullName = red\_str;

break;

case 2:

this->parents = red\_str;

break;

case 3:

this->spouses = red\_str;

break;

case 4:

this->children = red\_str;

break;

case 5:

this->dateBirh = red\_str;

break;

case 6:

this->dateDeath = red\_str;

break;

case 7:

this->age = red\_str;

break;

default:

cout << "Incorrect number of string!" << endl;

break;

}

}

Parents.h

#pragma once

//#include "libra.h"

#include "Person.h"

using namespace std;

class Parents : public Person {

private:

string fullName;

string parents;

string spouses;

string children;

string dateBirh;

string dateDeath;

string age;

public:

Parents();

~Parents();

Parents(const Parents& Parents\_copy);

Parents(string fullName);

string get\_fullName() const;

void set\_fullName(string fullName\_ch);

string get\_parents() const;

void set\_parents(string parents\_ch);

string get\_spouses() const;

void set\_spouses(string spouses\_ch);

string get\_children() const;

void set\_children(string children\_ch);

string get\_dateBirh() const;

void set\_dateBirh(string dateBirh\_ch);

string get\_dateDeath() const;

void set\_dateDeath(string dateDeath\_ch);

string get\_age() const;

void set\_age(string age\_ch);

virtual void show();

virtual void rewrite();

virtual void saving();

virtual void redact\_str(int num\_str, string red\_str);

};

Person.h

#pragma once

//#include "libra.h"

using namespace std;

class Person {

public:

Person() {};

virtual ~Person() {};

virtual void show() = 0;

virtual void rewrite() = 0;

virtual void saving() = 0;

virtual void redact\_str(int num\_str, string red\_str) = 0;

};

typedef struct El

{

Person\* data;

El\* next;

} El;

Keeper.h

#pragma once

//#include "libra.h"

#include "Person.h"

using namespace std;

class Keeper {

private:

int size = 0;

El\* head = NULL;

public:

Keeper();

Keeper(int size);

~Keeper();

Person\* operator[] (int id);

int get\_size();

void insert(Person\* new\_data);

void remove(int id);

void save();

void load();

};

Keeper.cpp

//#include "libra.h"

#include <iostream>

#include <fstream>

#include <string>

#include "Keeper.h"

#include "Children.h"

#include "Parents.h"

#include "GrandParents.h"

#include "You.h"

using namespace std;

Keeper::Keeper() : size(0), head(nullptr) {}

Keeper::Keeper(int size) : head(nullptr) { this->size = size; }

Keeper::~Keeper() {

if (head != NULL)

{

El\* buffer;

while (head->next != NULL)

{

buffer = head;

head = head->next;

buffer->data->~Person();

delete(buffer);

}

head->data->~Person();

delete(head);

}

}

Person\* Keeper::operator[] (const int id) {

if ((id >= size) || id < 0)

{

cout << "Incorrect index!" << endl;

exit(1);

}

else

{

El\* buffer = head;

for (int i = 0; i < size - id - 1; i++)

{

buffer = buffer->next;

}

return (buffer->data);

}

}

int Keeper::get\_size() { return size; }

void Keeper::insert(Person\* new\_data) {

El\* tmp;

tmp = new El;

if (size == 0)

{

tmp->data = new\_data;

tmp->next = 0;

++size;

}

else

{

tmp->data = new\_data;

tmp->next = head;

++size;

}

head = tmp;

}

void Keeper::remove(int id) {

if ((head != NULL) && (id < size) && (id >= 0))

{

El\* temp = head, \* temp2 = head;

for (int i = 0; i < id; i++)

{

temp2 = temp;

temp = temp->next;

}

if (temp == head)

{

head = temp->next;

}

else

{

temp2->next = temp->next;

}

temp->data->~Person();

delete(temp);

size--;

}

}

void Keeper::save() {

ofstream outfile;

string initfile = "data.txt";

outfile.open(initfile);

if (!outfile.is\_open())

{

cout << "Error opening file!" << endl;

exit(1);

}

else

{

outfile << size << endl;

outfile.close();

}

El\* buffer = head;

for (int i = 0; i < size; i++)

{

buffer->data->saving();

buffer = buffer->next;

}

}

void Keeper::load() {

if (size != 0)

{

El\* buffer;

while (head->next != NULL)

{

buffer = head;

head = head->next;

buffer->data->~Person();

delete(buffer);

}

head->data->~Person();

delete(head);

}

ifstream infile;

int read\_size, num\_person;

string a, b, c, d, e, f, g;

string initfile = "data.txt";

Person\* persons;

infile.open(initfile);

if (!infile.is\_open())

{

cout << "Error opening!" << endl;

exit(1);

}

infile >> read\_size;

for (int i = 0; i < read\_size; i++)

{

infile >> num\_person;

infile.ignore(32767, '\n');

if (num\_person == 1)

{

//infile >> a >> b >> c >> d >> e >> f >> g;

getline(infile, a);

getline(infile, b);

getline(infile, c);

getline(infile, d);

getline(infile, e);

getline(infile, f);

getline(infile, g);

Children\* children;

children = new Children;

children->set\_fullName(a);

children->set\_parents(b);

children->set\_spouses(c);

children->set\_children(d);

children->set\_dateBirh(e);

children->set\_dateDeath(f);

children->set\_age(g);

persons = children;

insert(persons);

}

if (num\_person == 2)

{

//infile >> a >> b >> c >> d >> e >> f >> g;

getline(infile, a);

getline(infile, b);

getline(infile, c);

getline(infile, d);

getline(infile, e);

getline(infile, f);

getline(infile, g);

Parents\* parents;

parents = new Parents;

parents->set\_fullName(a);

parents->set\_parents(b);

parents->set\_spouses(c);

parents->set\_children(d);

parents->set\_dateBirh(e);

parents->set\_dateDeath(f);

parents->set\_age(g);

persons = parents;

insert(persons);

}

if (num\_person == 3)

{

//infile >> a >> b >> c >> d >> e >> f >> g;

getline(infile, a);

getline(infile, b);

getline(infile, c);

getline(infile, d);

getline(infile, e);

getline(infile, f);

getline(infile, g);

GrandParents\* grandPar;

grandPar = new GrandParents;

grandPar->set\_fullName(a);

grandPar->set\_parents(b);

grandPar->set\_spouses(c);

grandPar->set\_children(d);

grandPar->set\_dateBirh(e);

grandPar->set\_dateDeath(f);

grandPar->set\_age(g);

persons = grandPar;

insert(persons);

}

if (num\_person == 4)

{

//infile >> a >> b >> c >> d >> e >> f >> g;

getline(infile, a);

getline(infile, b);

getline(infile, c);

getline(infile, d);

getline(infile, e);

getline(infile, f);

getline(infile, g);

You\* you;

you = new You;

you->set\_fullName(a);

you->set\_parents(b);

you->set\_spouses(c);

you->set\_children(d);

you->set\_dateBirh(e);

you->set\_dateDeath(f);

you->set\_age(g);

persons = you;

insert(persons);

}

}

}

Menu,h

#pragma once

//#include "libra.h"

#include <iostream>

#include <fstream>

#include "Keeper.h"

#include "Person.h"

using namespace std;

class Menu {

protected:

Keeper list;

Person\* persons;

string redact;

int exit = 1;

public:

void main\_menu();

int show\_options();

void show\_all\_persons();

void change\_data();

void add\_new();

void save\_to\_file();

void load\_from\_file();

void delete\_person();

void exit\_from\_programm();

void average();

};

Menu,cpp

//#include "libra.h"

#include <iostream>

#include <fstream>

#include <string>

#include "Menu.h"

#include "Children.h"

#include "Parents.h"

#include "GrandParents.h"

#include "You.h"

using namespace std;

void Menu::main\_menu() {

while (exit == 1)

{

switch (show\_options())

{

case 1:

show\_all\_persons();

break;

case 2:

change\_data();

break;

case 3:

add\_new();

break;

case 4:

save\_to\_file();

break;

case 5:

load\_from\_file();

break;

case 6:

delete\_person();

break;

default:

exit\_from\_programm();

break;

}

}

}

int Menu::show\_options() {

int c1;

cout << "\nWhat do you want to do?" << endl;

cout << "1 - Show all family member" << endl;

cout << "2 - Change family member's data" << endl;

cout << "3 - Add new family member" << endl;

cout << "4 - Save all changes to the file" << endl;

cout << "5 - Load data from the file" << endl;

cout << "6 - Delete a family member" << endl;

cout << "0 - Exit" << endl;

cout << "-> ";

cin >> c1;

return c1;

}

void Menu::show\_all\_persons()

{

try

{

if (list.get\_size() == 0)

{

throw "There's nothing to show.";

}

for (int i = 0; i < list.get\_size(); i++)

{

cout << "\_" << endl;

cout << i + 1 << "|";

list[i]->show();

}

}

catch (const char\* ex) { cout << ex << endl; }

}

void Menu::change\_data()

{

int c2, c3;

try

{

cout << "What human do you want to change (from 1 to " << list.get\_size() << ")?" <<

endl;

cin >> c2;

if ((c2 < 1) || (c2 > list.get\_size()))

{

throw "Incorrect index!";

}

list[c2 - 1]->show();

cout << "What string do you want to change?" << endl;

cout << "-> ";

cin >> c3;

cin.ignore(32767, '\n');

cout << "What do you want to put in this string?" << endl;

cout << "-> ";

if (c3 == 5)

{

bool fl = false;

while (fl == false)

{

try

{

getline(cin, redact);

//float f = stof(redact);

fl = true;

}

catch (...)

{

cout << "Enter the spouses" << endl;

}

}

list[c2 - 1]->redact\_str(c3, redact);

}

else

{

getline(cin, redact);

list[c2 - 1]->redact\_str(c3, redact);

cout << "The string was redacted." << endl;

}

}

catch (const char\* ex) { cout << ex << endl; }

}

void Menu::add\_new()

{

int c2;

cout << "\nWhat person do you want to add?" << endl;

cout << "1 - Childen" << endl;

cout << "2 - Parents" << endl;

cout << "3 - GrandParents" << endl;

cout << "4 - You" << endl;

cout << "0 - Go back" << endl;

cout << "-> ";

cin >> c2;

switch (c2)

{

case 1:

Children\* children;

children = new Children;

persons = children;

children->rewrite();

list.insert(persons);

cout << "The new person was added." << endl;

break;

case 2:

Parents\* parents;

parents = new Parents;

persons = parents;

parents->rewrite();

list.insert(persons);

cout << "The new person was added." << endl;

break;

case 3:

GrandParents \* grandPar;

grandPar = new GrandParents;

persons = grandPar;

grandPar->rewrite();

list.insert(persons);

cout << "The new human was added." << endl;

break;

case 4:

You \* you;

you = new You;

persons = you;

you->rewrite();

list.insert(persons);

cout << "The new human was added." << endl;

break;

default:

break;

}

}

void Menu::save\_to\_file()

{

try

{

if (list.get\_size() == 0)

{

throw "There's nothing to save.";

}

list.save();

cout << "The data was saved to the file." << endl;

}

catch (const char\* ex) { cout << ex << endl; }

}

void Menu::load\_from\_file()

{

try

{

list.load();

cout << "The data was loaded from file." << endl;

}

catch (const char\* ex) { cout << ex << endl; }

}

void Menu::delete\_person()

{

int c2;

cout << "What person do you want to change (from 1 to " << list.get\_size() << ")?" << endl;

for (int i = 0; i < list.get\_size(); i++)

{

cout << "\_" << endl;

cout << i + 1 << "|";

list[i]->show();

}

cout << "-> ";

cin >> c2;

c2 = list.get\_size() - c2 + 1;

try {

if ((c2 < 1) || (c2 > list.get\_size()))

{

throw "Incorrect spouses!";

}

list.remove(c2 - 1);

cout << "The chosen person was deleted." << endl;

}

catch (const char\* i)

{

cout << i << endl;

}

}

void Menu::exit\_from\_programm()

{

exit = 0;

}

Main.cpp

#include "Menu.h"

using namespace std;

int main()

{

Menu menu;

menu.main\_menu();

return 0;

}

1. **Результат работы программы**

После запуска программы перед нами появляется меню, где пользователь может выбрать интересующую операцию.

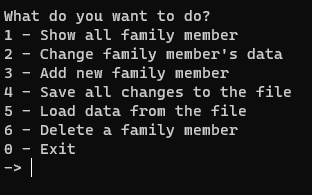


Рисунок 2 - Главное меню

Работа функции добавления члена семьи представлена на рисунке 3. Добавим себя:

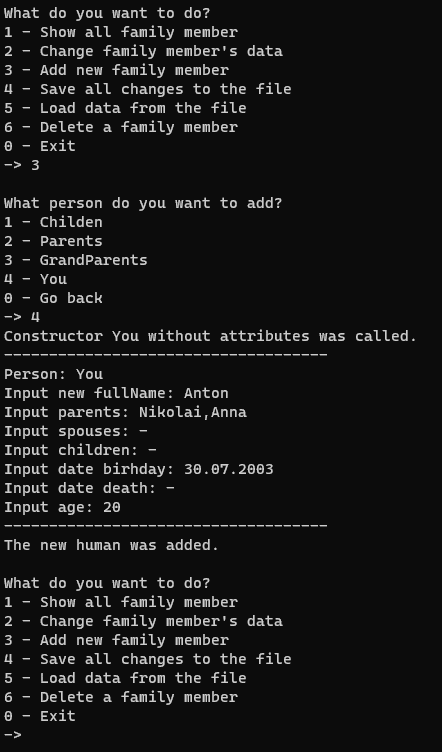


Рисунок 3 – Добавление себя

Занесем данные в файл.

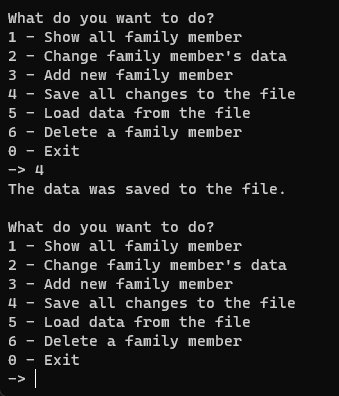


Рисунок 4 – Сохранение данных в файл

Закроем программу и запустим вновь. Выгрузим сохраненного машину из файла.

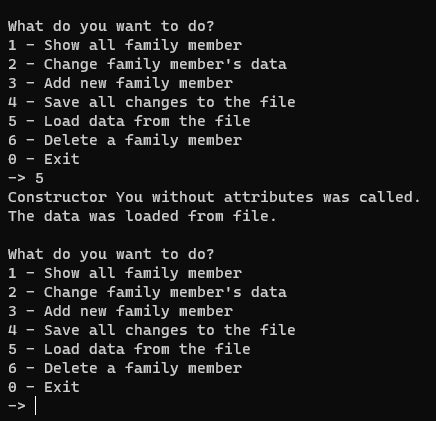


Рисунок 5 – Загрузка данных из файла

Изменим данные о вас:

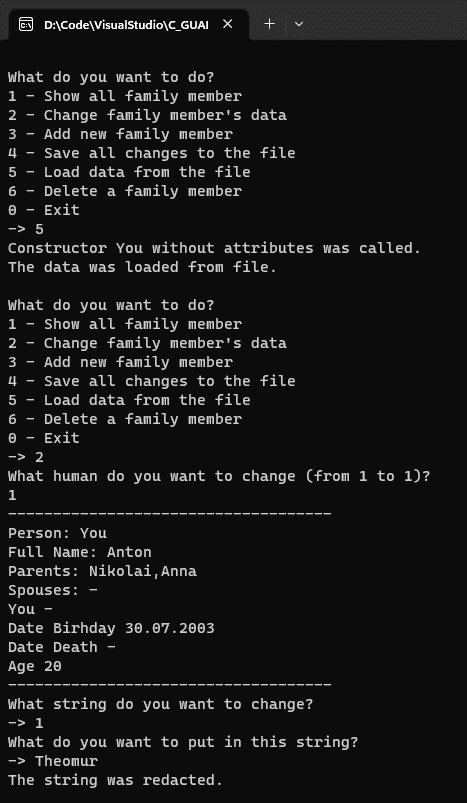


Рисунок 6 – Редактирование данных члена семьи

Удалим члена семьи:

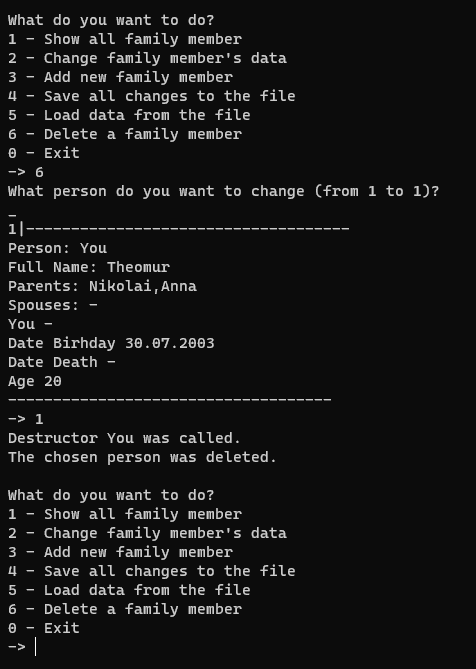


Рисунок 7 – Удаление члена семьи

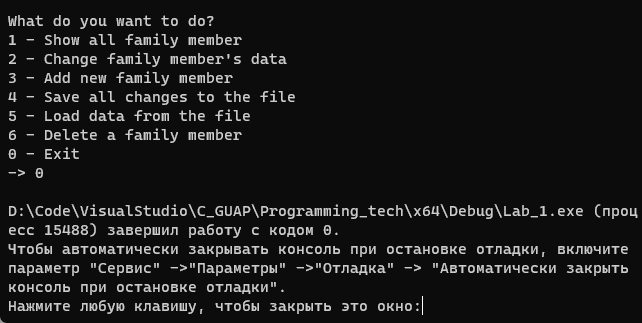


Рисунок 9 – Выход из программы

1. **Вывод**

В результате выполнения лабораторной работы был разработан класс Keeper, который реализует контейнер для хранения и обработки объектов. И имеет следующие функции: добавление и удаление производных объектов абстрактного класса Person, полное сохранение себя в файле, полное восстановление себя из файла. Были реализованы потомки класса Person: You, Children, GrandParents и Parents с реализацией интерфейса. В процессе выполнения лабораторной работы мы изучили основы применения контейнеров, вспомнили наследование классов, объектно-ориентированное программирование. Изучили принципы работы контейнеров, продемонстрировали их применение через создание указателя родительского класса на объект дочернего класса. На основе проведенных тестов можно сказать, что работы была выполнена успешно.

Ссылка на GitHub: [Theomur/Programming\_tech: задания 5 семестра по предмету технологии программирования (github.com)](https://github.com/Theomur/Programming_tech)