Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Инженерно-экономический факультет

Кафедра экономической информатики

ОТЧЕТ

по лабораторной работе

**«НАСЛЕДОВАНИЕ. ПРОСТОЕ**

**НАСЛЕДОВАНИЕ в языке C++»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 272303 |  | Н. Д. Вдовенко |
| Проверила |  | К.А.Борщевич |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Минск 2023

# Общая постановка задачи

**Цель работы:** изучить понятие базовых и производных классов, основные правила построения иерархии классов, особенности вызова конструкторов деструкторов при наследовании, механизмы разграничения доступа.

2. Создать класс «Человек», содержащий ФИО, дату рождения. Создать классы «Преподаватель университета» с полями «должность», «ученая степень», «специальность», «список научных трудов»; «Представитель комиссии» с полями «наименование комиссии», «год назначения в комиссию»; «Преподаватель-представитель комиссии» с полем «список работ, выполненных в комиссии». Классы должны содержать методы получения и изменения значений всех полей. Создать массив объектов базового класса и заполнить этот массив объектами базового и производных классов. Написать программу, которая выводит на экран и в файл данные только о тех преподавателях, которые имеют ученую степень. Классы должны содержать методы получения и изменения значений всех полей. Все поля классов должны быть объявлены с атрибутами private или protected.

**Ответы на контрольные вопросы**

1. В языке C++ есть три модификатора доступа к содержимому класса:  
public: позволяет доступ к элементам класса из любого места программы.  
private: ограничивает доступ к элементам класса только внутри самого класса.  
protected: разрешает доступ к элементам класса из самого класса и его производных классов.  
  
2. Цель использования наследования в объектно-ориентированном программировании заключается в повторном использовании кода, создании иерархии классов и специализации функциональности. Наследование позволяет производным классам наследовать свойства и методы базового класса, а также добавлять новые свойства и методы.  
  
3. При наследовании класс, от которого происходит наследование, называется базовым классом или суперклассом, а класс, который наследует свойства базового класса, называется производным классом или подклассом. Также базовый класс может называться родительским классом, а производный класс - дочерним классом.  
  
4. Различие между public, private и protected наследованием классов заключается в области видимости унаследованных элементов класса:  
public наследование делает все публичные элементы базового класса публичными в производном классе.  
private наследование делает все элементы базового класса приватными в производном классе, тем самым скрывая их от внешнего доступа.  
protected наследование делает все элементы базового класса защищенными в производном классе, что позволяет доступ к ним из производного класса и его потомков, но не извне.  
  
5. При создании объекта дочернего класса сначала вызывается конструктор базового класса, затем конструктор дочернего класса. Порядок вызова конструкторов соответствует иерархии наследования от базового класса до дочернего класса.  
  
6. При разрушении объекта дочернего класса сначала вызывается деструктор дочернего класса, а затем деструктор базового класса. Порядок вызова деструкторов соответствует обратной иерархии наследования от дочернего класса до базового класса.  
  
7. Отношение агрегации означает, что один класс содержит другой класс в качестве своего члена, но существует независимость между ними. Класс-контейнер хранит ссылку или указатель на класс-член, но класс-член может существовать независимо от класса-контейнера. Например, класс "Университет" может содержать классы "Преподаватель" и "Студент", но они могут существовать и без самого университета.

Отношение композиции означает, что один класс является составной частью другого класса и не может существовать независимо от него. Класс-контейнер полностью управляет жизненным циклом класса-члена и отвечает за его создание, хранение и удаление. Например, класс "Автомобиль" может содержать класс "Двигатель", и двигатель не имеет смысла без автомобиля.  
  
8. Пример наследования по произвольной предметной области с иерархией, содержащей 3 и более классов:  
  
class Shape {

protected:

int x, y;

public:

virtual void draw() = 0;

};

class Circle : public Shape {

private:

int radius;

public:

void draw() override {

// реализация для рисования круга

}

};

class Rectangle : public Shape {

private:

int width, height;

public:

void draw() override {

// реализация для рисования прямоугольника

}

};

class Triangle : public Shape {

private:

int side1, side2, side3;

public:

void draw() override {

// реализация для рисования треугольника

}

};

9. Дочерний класс не может унаследовать приватные (private) поля и методы базового класса. Приватные элементы класса доступны только внутри самого класса и недоступны в производных классах.  
  
10. Конструкторы и деструкторы наследуются в производных классах. При создании объекта производного класса сначала вызывается конструктор базового класса, а затем конструктор производного класса. При разрушении объекта производного класса сначала вызывается деструктор производного класса, а затем деструктор базового класса.

# Листинг

#pragma once

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

int checkInt(int a);

class Person {

private:

string name;

string birthDate;

public:

Person(const string& name, const string& birthDate)

: name(name), birthDate(birthDate) {}

string getName() const {

return name;

}

string getBirthDate() const {

return birthDate;

}

virtual bool isUniversityTeacher() const {

return false;

}

virtual void printDetails(ostream& out) const = 0;

virtual void printDetails(ofstream& outFile) const = 0;

};

class UniversityTeacher : public Person {

private:

string position;

string academicDegree;

string specialization;

string\* scientificWorks;

int numScientificWorks;

public:

UniversityTeacher(const string& name, const string& birthDate,

const string& position, const string& academicDegree,

const string& specialization,

const string\* scientificWorks, int numScientificWorks)

: Person(name, birthDate), position(position), academicDegree(academicDegree),

specialization(specialization), scientificWorks(new string[numScientificWorks]),

numScientificWorks(numScientificWorks) {

for (int i = 0; i < numScientificWorks; i++) {

this->scientificWorks[i] = scientificWorks[i];

}

}

~UniversityTeacher() {

delete[] scientificWorks;

}

string getPosition() const {

return position;

}

string getAcademicDegree() const {

return academicDegree;

}

string getSpecialization() const {

return specialization;

}

string\* getScientificWorks() const {

return scientificWorks;

}

int getNumScientificWorks() const {

return numScientificWorks;

}

bool isUniversityTeacher() const override {

return true;

}

void printDetails(ostream& out) const override {

if (!getAcademicDegree().empty()) {

out << "Name: " << getName() << endl;

out << "Position: " << getPosition() << endl;

out << "Academic Degree: " << getAcademicDegree() << endl;

out << "Specialization: " << getSpecialization() << endl;

out << "Scientific Works:" << endl;

string\* scientificWorks = getScientificWorks();

int numScientificWorks = getNumScientificWorks();

for (int j = 0; j < numScientificWorks; j++) {

out << "- " << scientificWorks[j] << endl;

}

out << "------------------------------" << endl;

}

}

void printDetails(ofstream& outFile) const override {

if (!getAcademicDegree().empty()) {

outFile << "Name: " << getName() << endl;

outFile << "Position: " << getPosition() << endl;

outFile << "Academic Degree: " << getAcademicDegree() << endl;

outFile << "Specialization: " << getSpecialization() << endl;

outFile << "Scientific Works:" << endl;

string\* scientificWorks = getScientificWorks();

int numScientificWorks = getNumScientificWorks();

for (int j = 0; j < numScientificWorks; j++) {

outFile << "- " << scientificWorks[j] << endl;

}

outFile << "------------------------------" << endl;

}

}

};

class CommissionMember : public Person {

private:

string commissionName;

int appointmentYear;

public:

CommissionMember(const string& name, const string& birthDate, const string& commissionName, int appointmentYear)

: Person(name, birthDate), commissionName(commissionName), appointmentYear(appointmentYear) {}

string getCommissionName() const {

return commissionName;

}

int getAppointmentYear() const {

return appointmentYear;

}

void printDetails(ostream& out) const override {

out << "Name: " << getName() << endl;

out << "Commission Name: " << getCommissionName() << endl;

out << "Appointment Year: " << getAppointmentYear() << endl;

out << "------------------------------" << endl;

}

void printDetails(ofstream& outFile) const override {

outFile << "Name: " << getName() << endl;

outFile << "Commission Name: " << getCommissionName() << endl;

outFile << "Appointment Year: " << getAppointmentYear() << endl;

outFile << "------------------------------" << endl;

}

};

class CommissionTeacher : public UniversityTeacher {

private:

string\* commissionWorks;

int numCommissionWorks;

public:

CommissionTeacher(const string& name, const string& birthDate,

const string& position, const string& academicDegree,

const string& specialization,

const string\* scientificWorks, int numScientificWorks,

const string\* commissionWorks, int numCommissionWorks) :

UniversityTeacher(name, birthDate, position, academicDegree, specialization, scientificWorks, numScientificWorks),

commissionWorks(new string[numCommissionWorks]), numCommissionWorks(numCommissionWorks) {

for (int i = 0; i < numCommissionWorks; i++) {

this->commissionWorks[i] = commissionWorks[i];

}

}

~CommissionTeacher() {

delete[] commissionWorks;

}

string\* getCommissionWorks() const {

return commissionWorks;

}

int getNumCommissionWorks() const {

return numCommissionWorks;

}

void printDetails(ostream& out) const override {

UniversityTeacher::printDetails(out);

out << "Commission Works:" << endl;

string\* commissionWorks = getCommissionWorks();

int numCommissionWorks = getNumCommissionWorks();

for (int j = 0; j < numCommissionWorks; j++) {

out << "- " << commissionWorks[j] << endl;

}

out << "------------------------------" << endl;

}

void printDetails(ofstream& outFile) const override {

UniversityTeacher::printDetails(outFile);

outFile << "Commission Works:" << endl;

string\* commissionWorks = getCommissionWorks();

int numCommissionWorks = getNumCommissionWorks();

for (int j = 0; j < numCommissionWorks; j++) {

outFile << "- " << commissionWorks[j] << endl;

}

outFile << "------------------------------" << endl << endl;

}

};  
  
  
#include "header.h"

int checkInt(int a) {

while (true) {

cin >> a;

if (cin.fail()) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "The input can only contain numbers. Try again." << endl;

continue;

}

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

if (cin.gcount() > 1) {

cout << "The input can only contain numbers. Try again." << endl;

continue;

}

if (a < 0) {

cout << "The input cannot contain negative numbers. Try again." << endl;

continue;

}

break;

}

return a;

}

#include "header.h"

int main() {

const int MAX\_TEACHERS = 100;

Person\*\* teachers = new Person \* [MAX\_TEACHERS];

int numTeachers = 0;

ofstream outFile("teachers.txt", ios::app);

int choice = 0;

do {

cout << "Menu:" << endl;

cout << "1. Add University Teacher" << endl;

cout << "2. Add Commission Teacher" << endl;

cout << "3. Add Commission Member" << endl;

cout << "4. Show Teachers" << endl;

cout << "5. Exit" << endl;

cout << "Enter your choice: ";

choice = checkInt(choice);

switch (choice) {

case 1: {

string name, birthDate, position, academicDegree, specialization;

int numScientificWorks = 0;

cout << "Enter name: ";

cin >> name;

cout << "Enter birth date: ";

cin >> birthDate;

cout << "Enter position: ";

cin >> position;

cout << "Enter academic degree: ";

cin >> academicDegree;

cout << "Enter specialization: ";

cin >> specialization;

cout << "Enter the number of scientific works: ";

numScientificWorks = checkInt(numScientificWorks);

string\* scientificWorks = new string[numScientificWorks];

cout << "Enter scientific works:" << endl;

for (int i = 0; i < numScientificWorks; i++) {

cout << "Scientific work " << i + 1 << ": ";

cin >> scientificWorks[i];

}

teachers[numTeachers] = new UniversityTeacher(name, birthDate, position, academicDegree, specialization, scientificWorks, numScientificWorks);

numTeachers++;

delete[] scientificWorks;

break;

}

case 2: {

string name, birthDate, position, academicDegree, specialization;

int numScientificWorks = 0, numCommissionWorks = 0;

cout << "Enter name: ";

cin >> name;

cout << "Enter birth date: ";

cin >> birthDate;

cout << "Enter position: ";

cin >> position;

cout << "Enter academic degree: ";

cin >> academicDegree;

cout << "Enter specialization: ";

cin >> specialization;

cout << "Enter the number of scientific works: ";

numScientificWorks = checkInt(numScientificWorks);

string\* scientificWorks = new string[numScientificWorks];

cout << "Enter scientific works:" << endl;

for (int i = 0; i < numScientificWorks; i++) {

cout << "Scientific work " << i + 1 << ": ";

cin >> scientificWorks[i];

}

cout << "Enter the number of commission works: ";

numCommissionWorks = checkInt(numCommissionWorks);

string\* commissionWorks = new string[numCommissionWorks];

cout << "Enter commission works:" << endl;

for (int i = 0; i < numCommissionWorks; i++) {

cout << "Commission work " << i + 1 << ": ";

cin >> commissionWorks[i];

}

teachers[numTeachers] = new CommissionTeacher(name, birthDate, position, academicDegree, specialization, scientificWorks, numScientificWorks, commissionWorks, numCommissionWorks);

numTeachers++;

delete[] scientificWorks;

delete[] commissionWorks;

break;

}

case 3: {

string name, birthDate, commissionName;

int appointmentYear = 0;

cout << "Enter name: ";

cin >> name;

cout << "Enter birth date: ";

cin >> birthDate;

cout << "Enter commission name: ";

cin >> commissionName;

cout << "Enter appointment year: ";

appointmentYear = checkInt(appointmentYear);

teachers[numTeachers] = new CommissionMember(name, birthDate, commissionName, appointmentYear);

numTeachers++;

break;

}

case 4: {

if (numTeachers == 0) {

cout << "No teachers to show." << endl;

}

else {

for (int i = 0; i < numTeachers; i++) {

if (teachers[i]->isUniversityTeacher()) {

teachers[i]->printDetails(cout);

}

}

}

break;

}

case 5:

{

break;

}

default:

{

cout << "Invalid choice. Please try again." << endl;

}

}

} while (choice != 5);

for (int i = 0; i < numTeachers; i++) {

if (teachers[i]->isUniversityTeacher()) {

teachers[i]->printDetails(outFile);

}

}

for (int i = 0; i < numTeachers; i++) {

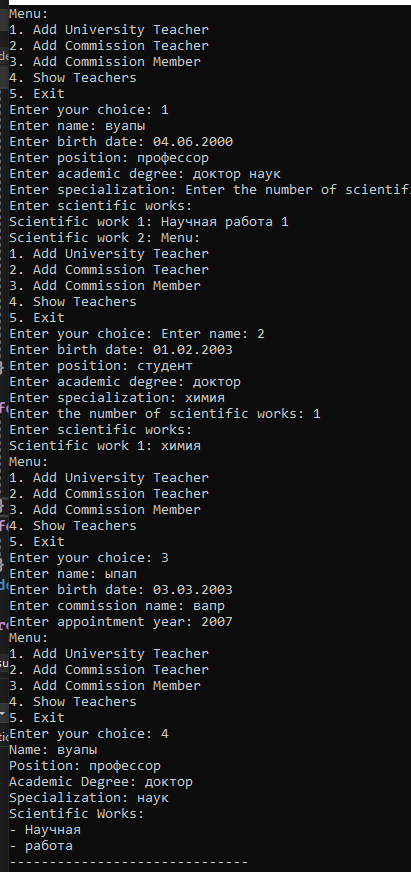
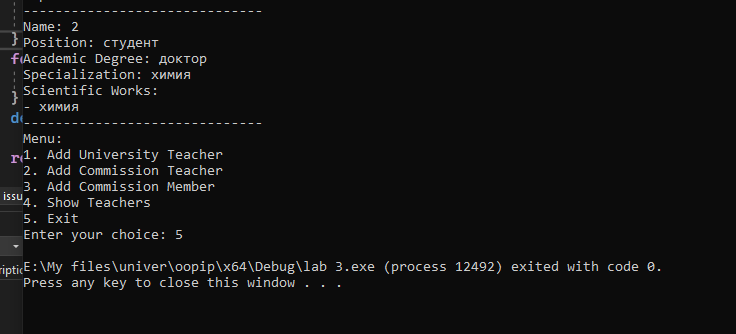
delete teachers[i];

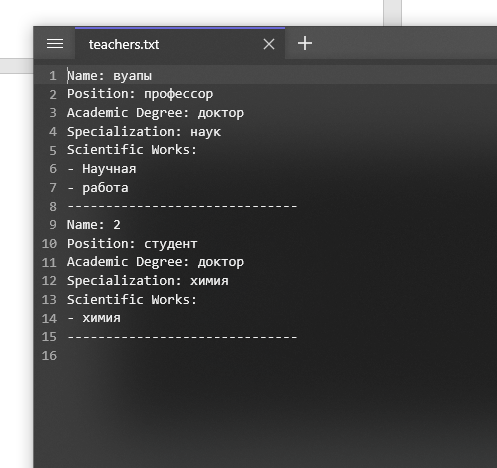
}

delete[] teachers;

return 0;

}



# Выводы

Изучил базовые и производные классы, правила иерархии классов, вызов конструкторов и деструкторов при наследовании и механизмы разграничения доступа.