# 面向对象程序设计过程学习考核报告

题目 师生信息管理程序设计

学院系别 经济管理学院管理工程系

学年学期 2023-2024（2）

课程名称 面向对象程序设计

任课教师 杜茂康

专业年级 信息管理与信息系统 2023 级学生班级 03012303

学生姓名 丁同勖

学生学号 2023211281

完成日期 2024年6月11日

考核成绩

1. 过程学习题目分析(10分)

题目要求设计一个教学系统来管理不同类型的人员（包括教师和学生）的档案信息，并实现各种操作，如数据输入、输出、查询、删除，以及导师指导的学生信息管理。具体需求包括：

1. 创建多个类以表示不同类型的人员，包括普通教师、院长、系主任和学生。
2. 各类人员的档案信息要能够输入和输出。
3. 管理导师与学生之间的指导关系。
4. 以Person为接口访问各类的输入、输出和工资管理等虚函数，实现多态。
5. 使用STL容器来管理和操作这些对象。

### 解决问题的思路

1. **抽象类和派生类**：
   * 抽象出 Person 类作为基类，表示所有人员的公共属性和方法。
   * 从 Person 派生出 Teacher 和 Student 类，分别表示教师和学生，并扩展各自的特有属性和方法。
   * 从 Teacher 派生出 OrdinaryTeacher、Dean 和 President 类，表示不同级别的教师，并实现各自特有的收入计算方法。
2. **处理类之间的关系**：
   * 使用继承来实现类的层次结构，使得 Student 和 Teacher 共享 Person 类的属性和方法。
   * 使用组合关系，Student 类中包含一个指向 Teacher 对象的指针，用于表示导师。
   * 在 Teacher 类中包含一个学生列表，记录其指导的所有学生。
3. **存储对象数据**：
   * 使用STL中的 std::vector 容器来存储所有 Person 对象的指针，便于统一管理和遍历。
   * 使用 std::map 容器来管理导师与学生的关系，键为 Teacher 指针，值为其指导的 Student 列表。

### 抽象出的类

1. **Person 类**：
   * 数据成员：姓名、年龄、性别
   * 成员函数：输入数据、输出数据、虚函数接口
2. **Teacher 类（派生自 Person）**：
   * 数据成员：教师编号、职称、系别、基本工资
   * 成员函数：输入数据、输出数据、设置工资、获取工资、纯虚函数 getIncome 表示总收入计算
3. **OrdinaryTeacher 类（派生自 Teacher）**：
   * 数据成员：课时费、课时数
   * 成员函数：输入数据、输出数据、计算总收入
4. **Dean 类（派生自 Teacher）**：
   * 数据成员：绩效工资
   * 成员函数：输入数据、输出数据、计算总收入
5. **President 类（派生自 Teacher）**：
   * 数据成员：年薪
   * 成员函数：输入数据、输出数据、计算总收入
6. **Student 类（派生自 Person）**：
   * 数据成员：学号、班级、各科成绩、导师指针
   * 成员函数：输入数据、输出数据、设置导师、获取导师

### 类之间的关系处理

* **继承关系**：
  + Person 类是基类，Teacher 和 Student 类继承自 Person。
  + Teacher 类派生出 OrdinaryTeacher、Dean 和 President 类。
* **组合关系**：
  + 在 Student 类中添加 Teacher\* mentor，表示导师。
  + 在 Teacher 类中使用 std::vector<Student\*> 列表，记录其指导的学生。

### 存储对象数据的方法

* 使用 std::vector<Person\*> 来存储所有 Person 对象的指针，便于统一管理和操作。
* 使用 std::map<Teacher\*, std::vector<Student\*>> 来存储导师与学生的关系，键为 Teacher 指针，值为其指导的学生列表。

### 代码实现

具体的代码实现中，主要包含以下操作：

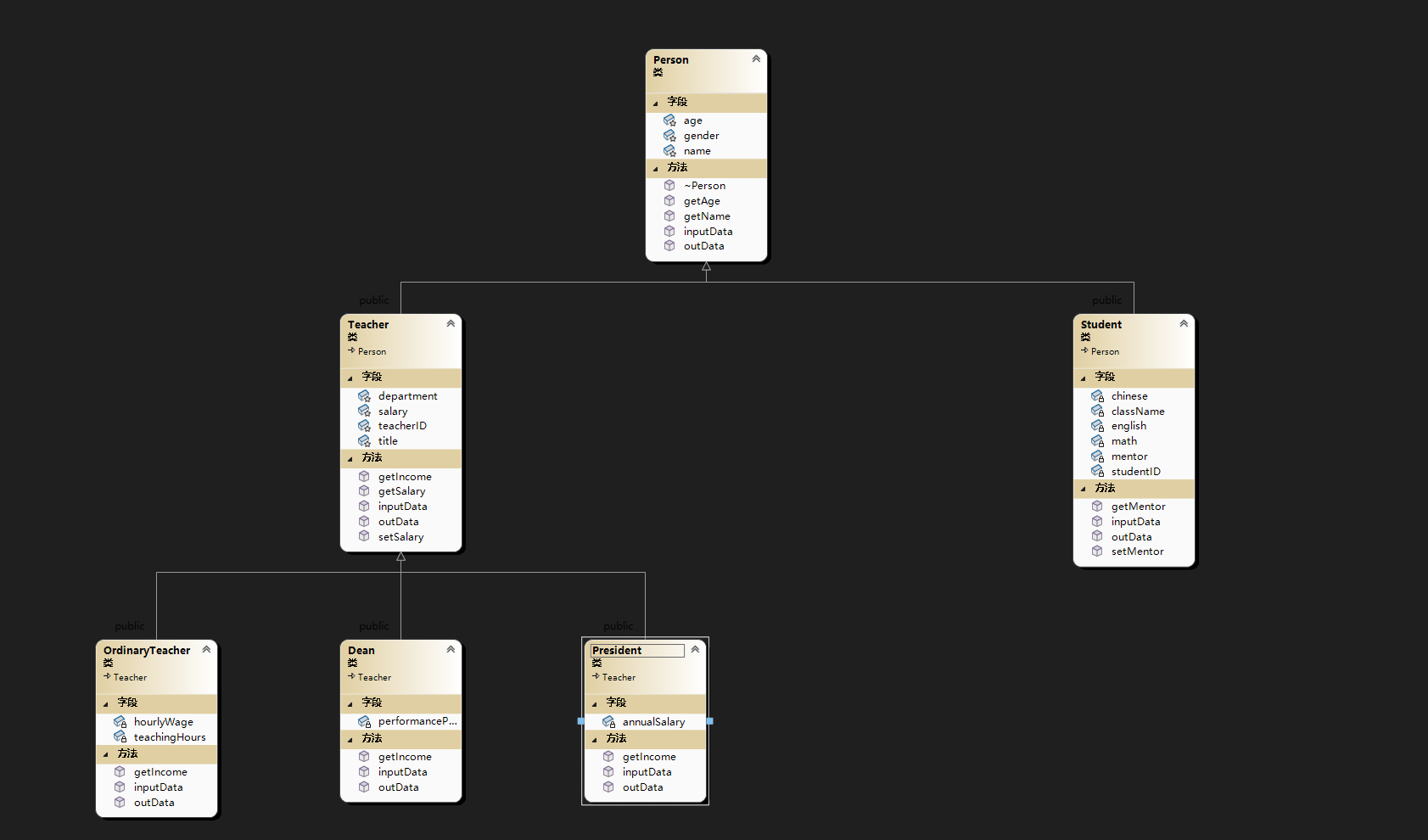
* 添加人员：根据用户选择的类型创建相应对象并存储到 persons 容器中。
* 显示所有人员：遍历 persons 容器并调用各对象的 outData 方法。
* 查询人员：通过姓名查找并显示对应对象的信息。
* 删除人员：通过姓名删除对象并同步更新导师和学生的关系。
* 管理学生的导师信息：增加、删除或更改学生的导师。
* 管理教师的学生信息：增加或删除教师指导的学生。

1. 类设计方案（20分）

### 解决方案概述

1. **分析对象和抽象类**：
   * 我们需要管理教师和学生两类对象。为了体现对象的层次结构，首先抽象出一个基类 Person，用于表示所有人员的公共属性和方法。
   * 从 Person 派生出 Teacher 和 Student 类，分别表示教师和学生，并扩展各自特有的属性和方法。
   * 为了进一步区分不同级别的教师，从 Teacher 派生出 OrdinaryTeacher、Dean 和 President 类，表示普通教师、系主任和院长。
2. **类之间的关系**：
   * 使用继承来实现类的层次结构，使 Student 和 Teacher 共享 Person 类的属性和方法。
   * 使用组合关系：Student 类包含一个指向 Teacher 对象的指针，用于表示导师；Teacher 类包含一个 std::vector<Student\*>，记录其指导的所有学生。
3. **对象数据的存储**：
   * 使用 std::vector<Person\*> 容器来存储所有 Person 对象的指针，便于统一管理和遍历。
   * 使用 std::map<Teacher\*, std::vector<Student\*>> 容器来管理导师与学生的关系，键为 Teacher 指针，值为其指导的学生列表。

### UML 类图



### 各类的概要模型

#### 类 Person

class Person {

protected:

std::string name;

int age;

std::string gender;

public:

virtual void inputData();

virtual void outputData();

virtual void setSalary(double salary) = 0;

virtual double getSalary() const = 0;

virtual double getIncome() const = 0;

virtual ~Person() = default;

};

#### 类 Teacher

class Teacher : public Person {

protected:

std::string id;

std::string title;

std::string dept;

double salary;

std::vector<Student\*> students;

public:

virtual void inputData() override;

virtual void outputData() override;

virtual void setSalary(double salary) override;

virtual double getSalary() const override;

virtual double getIncome() const override = 0;

};

#### 类 OrdinaryTeacher

class OrdinaryTeacher : public Teacher {

private:

double hourlyPay;

int hours;

public:

virtual void inputData() override;

virtual void outputData() override;

virtual double getIncome() const override;

};

#### 类 Dean

class Dean : public Teacher {

private:

double bonus;

public:

virtual void inputData() override;

virtual void outputData() override;

virtual double getIncome() const override;

};

#### 类 President

class President : public Teacher {

private:

double annualPay;

public:

virtual void inputData() override;

virtual void outputData() override;

virtual double getIncome() const override;

};

#### 类 Student

class Student : public Person {

private:

std::string studentID;

std::string className;

std::map<std::string, double> grades;

Teacher\* mentor;

public:

virtual void inputData() override;

virtual void outputData() override;

void setMentor(Teacher\* mentor);

Teacher\* getMentor() const;

};

### 主要方法介绍

* **inputData()**：用于输入对象的详细信息，覆盖不同类的特定需求。
* **outputData()**：用于输出对象的详细信息，覆盖不同类的特定需求。
* **setSalary() 和 getSalary()**：设置和获取基本工资。
* **getIncome()**：纯虚函数，用于计算不同类型教师的总收入。
* **setMentor() 和 getMentor()**：设置和获取学生的导师。

### 数据存储

* 使用 std::vector<Person\*> 存储所有人员对象的指针。
* 使用 std::map<Teacher\*, std::vector<Student\*>> 存储导师与学生的关系。

这种设计通过类的继承和多态，实现了教师和学生信息的统一管理，方便了信息的输入、输出、查询、删除以及导师与学生之间的指导关系管理。

1. 程序调测过程中的问题和解决方法（20分）

详细记录程序在C++环境中编辑、调试过程中遇到的问题。可以用日志的方式描述，细到语法错误，大到设计不足的错误（比如，没有设计无参构造函数致使无法定义对象数组，没有定义复制构造函数致使对象复制产生指针悬挂问题，没有重载类的operator<运算符，导致使用STL Vector或list排序对象时出现不能添加或输出对象的错误），越详细越好！

可以用类似于下面的方式进行本部分的记录：

|  |  |
| --- | --- |
| 问题1 | 通过姓名删除老师时，无法做到删除老师的时候同步删除他学生的指导老师信息，删除学生的时候同步删除他老师的指导学生信息 |
| 解决方法 | deletePersonByName 函数被修改为：如果删除的是老师，会清空该老师指导的所有学生的导师信息。如果删除的是学生，会从导师的学生列表中删除该学生的信息。确保删除人员时导师和学生信息在任何时候都是一致的。 |
| 问题2 | 通过姓名删除老师时，没有同步将学生类储存的指导教师指针置空，输出全体成员时报错。 |
| 解决方法 | 删除老师时将学生类储存的指导教师指针置空。 |
| 问题3 | 设计了通过学生调整指导老师时，一个学生只能设置一个指导老师的限制。但忘记设置通过老师调整指导的学生时，学生只能有一个指导老师的限制。 |
| 解决方法 | 在执行void manageTeachersStudents(…) 函数时，如果学生已经有指导老师则直接替换而不是增加指导老师，并输出进行了替换操作的提示。 |
| 问题4 | 从学生管理指导老师和从老师管理指导学生，难以同步更新，导致输出老师指导的学生和学生的指导老师信息不一致。 |
| 解决方法 | 通过std::map<Teacher\*, std::vector<Student\*>> teacherStudentMap;管理学生和指导老师的关系，做到同步更新。并且增加了通过教师姓名调整他所指导的学生信息（manageTeachersStudents 函数），并且调整了manageStudentMentor函数来同步更新学生和教师的信息。这样用户可以选择添加或删除学生的导师，或者根据学生名字管理导师信息，同时保持数据的一致性。 |
| 问题5 | 没做到输出教师信息的时候同步输出所有他指导的学生。 |
| 解决方法 | **一开始的方法**是在 Teacher 类的重载输出运算符方法中添加对学生信息的输出。为每个 Teacher 类对象添加一个成员来存储其指导的学生列表，然后在输出信息时一并显示这些学生的信息。  **但后续觉得**存储两次教师学生之间的指导关系浪费系统资源，就想到可以通过 std::map<Teacher\*,std::vector<Student\*>>& teacherStudentMap 来管理教师和学生之间的关系，并在输出教师信息时从 teacherStudentMap获取其指导的学生信息。就从基类一直到所有派生类都添加一个了outData函数，使其能接受const std::map<Teacher\*, std::vector<Student\*>>& teacherStudentMap参数，在执行输出成员时传递teacherStudentMap参数以供处理输出。 |
| 问题6 | 按同类对象通过年龄排序出错，无法正确排序及输出。 |
| 解决方法 | 新增classifyAndSortPersonsByAge 函数，首先将 persons 中的对象按类型分类到不同的容器中。然后，对每一类对象的容器分别按年龄进行排序。  最后，输出排序后的结果，分别输出学生、普通教师、系主任和院长的信息，确保每类对象都按年龄排序。 |
| 问题7 | Teacher和Student类交叉引用，报错：C2027:使用了未定义类型 |
| 解决方法 | 加上前向引用class Student; // 前向声明后可正常运行。 |
| 问题8 | 但随后由于程序修改完善，实现通过学生管理指导教师和通过教师管理所指导的学生功能时，Teacher和Student类出现了双向交叉引用，通过简单的添加前向声明已经无法解决。 |
| 解决方法 | 进行大改版，使用预处理指令include，将单独编译的文件引入到预处理命令include的地方，将不同的类分隔成不同的类文件，不同的类将类声明和类定义分隔开，在需要使用其他类时引入目标类的头文件（引入声明）即可使用目标类。通过单独编译和向前声明方法实现两个类之间相互引用成员。（第一次这么操作，没有任何经验，对着报错慢慢改了很久） |
| 问题9 | 在使用 vector 和 map 时由于不熟练，频繁出现语法错误。 |
| 解决方法 | 核对书本，仔细修改校对，并加深印象。 |
| 问题10 | 在使用switch…case…语句时，没有做选择范围之外值的处理，导致输入预期外的值后程序无法继续运行。 |
| 解决方法 | 添加default:处理预期外的值。 |
| 问题11 | 在我的程序中，单纯简单重载输出运算符没有办法实现我所需要的功能，解决问题 5 所使用的解决方案需要输出能接受接受传递std::vector<Student\*>>& teacherStudentMap类型参数来输出老师和学生之间的指导关系。 |
| 解决方法 | 从基类一直到所有派生类都添加一个了outData函数，使其能接受const std::map<Teacher\*, std::vector<Student\*>>& teacherStudentMap参数。并修改重载输出运算符，使其调用outData函数（保留了Person类的重载输出运算符，虽然已经不再使用该方法输出信息） |
| 问题12 | 在设计分类排序函数classifyAndSortPersonsByAge()时，输出结果时忘记设置给outData函数传递teacherStudentMap参数，导致输出时无法输出老师指导的学生和学生的指导老师。 |
| 解决方法 | classifyAndSortPersonsByAge()函数调用outData函数时传入teacherStudentMap参数 |
| 问题13 | 将不同的类分隔成不同的类文件，不同的类将类声明和类定义分隔开时，编译器报错类型重定义。 |
| 解决方法 | 在头文件中使用#ifndef #define #endif防止类型重定义。 |
| 问题14 | 对Student.cpp和Teacher.cpp报错不允许指针指向不完整的类类型。 |
| 解决方法 | 在Student.cpp和Teacher.cpp开头添加  #include "Student.h"和#include "Teacher.h" |

四、程序源码（40分）

列出程序的所有源码，包括每个类的头文件和源码文件，主程序文件（包含main的文件），以文件内容为单位列出全部源码。形式如下：

Person.h

#ifndef PERSON\_H

#define PERSON\_H

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

#include <map>

//#include "Teacher.h"

//#include "Student.h"

class Student; // 前向声明

class Teacher;

class Person {

protected:

std::string name;

int age;

std::string gender;

public:

virtual void inputData();

virtual void outData(const std::map<Teacher\*, std::vector<Student\*>>& teacherStudentMap) const;

virtual ~Person() = default;

friend std::ostream& operator<<(std::ostream& os, const Person& p);

const std::string& getName() const;

int getAge() const;

};

#endif // PERSON\_H

Person.cpp

#include "Person.h"

void Person::inputData() {

std::cout << "\n请输入姓名: ";

std::cin >> name;

std::cout << "请输入年龄: ";

std::cin >> age;

std::cout << "请输入性别: ";

std::cin >> gender;

}

void Person::outData(const std::map<Teacher\*, std::vector<Student\*>>& teacherStudentMap) const {

std::cout << "\n==============详细信息==============" << "\n姓名: " << name << "\n年龄: " << age << "\n性别: " << gender;

}

std::ostream& operator<<(std::ostream& os, const Person& p) {

p.outData(std::map<Teacher\*, std::vector<Student\*>>());

return os;

}

const std::string& Person::getName() const {

return name;

}

int Person::getAge() const {

return age;

}

Student.h

#ifndef STUDENT\_H

#define STUDENT\_H

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

#include <map>

#include "Person.h"

//#include "Teacher.h"

class Teacher; // 前向声明

//class Person;

class Student : public Person {

private:

std::string studentID;

std::string className;

double chinese;

double math;

double english;

Teacher\* mentor; // 指导老师

public:

void inputData() override;

void outData(const std::map<Teacher\*, std::vector<Student\*>>& teacherStudentMap) const override;

void setMentor(Teacher\* mentor);

Teacher\* getMentor() const;

};

#endif

Student.cpp

#include "Student.h"

#include "Teacher.h"

void Student::inputData() {

Person::inputData();

std::cout << "输入学生学号: ";

std::cin >> studentID;

std::cout << "输入学生班级: ";

std::cin >> className;

std::cout << "输入学生语文成绩: ";

std::cin >> chinese;

std::cout << "输入学生数学成绩: ";

std::cin >> math;

std::cout << "输入学生英语成绩: ";

std::cin >> english;

}

void Student::outData(const std::map<Teacher\*, std::vector<Student\*>>& teacherStudentMap) const {

Person::outData(std::map<Teacher\*, std::vector<Student\*>>());

std::cout << "\n学生学号: " << studentID << "\n学生班级: " << className << "\n学生导师: ";

if (Student::mentor == nullptr)

std::cout << "无\n";

else

std::cout << mentor->getName() << std::endl;

std::cout << "学生语文成绩: " << chinese << "\n学生数学成绩: " << math << "\n学生英语成绩: " << english << std::endl;

}

void Student::setMentor(Teacher\* mentor) {

this->mentor = mentor;

}

Teacher\* Student::getMentor() const {

return mentor;

}

Teacher.h

#ifndef TEACHER\_H

#define TEACHER\_H

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

#include <map>

//#include "Student.h"

#include "Person.h"

class Student; // 前向声明

//class Person;

class Teacher : public Person {

protected:

std::string teacherID;

std::string title;

std::string department;

double salary;

public:

void inputData() override;

void outData(const std::map<Teacher\*, std::vector<Student\*>>& teacherStudentMap) const override;

virtual double getSalary() const;

virtual void setSalary(double salary);

virtual double getIncome() const = 0; // 纯虚函数

friend std::ostream& operator<<(std::ostream& os, const Teacher& t);

};

#endif

Teacher.cpp

#include "Teacher.h"

#include "Student.h"

void Teacher::inputData() {

Person::inputData();

std::cout << "输入教师编号: ";

std::cin >> teacherID;

std::cout << "输入教师职称: ";

std::cin >> title;

std::cout << "输入教师系别: ";

std::cin >> department;

std::cout << "输入教师基本工资: ";

std::cin >> salary;

}

void Teacher::outData(const std::map<Teacher\*, std::vector<Student\*>>& teacherStudentMap) const {

Person::outData(teacherStudentMap);

std::cout << "\n教师编号: " << teacherID << "\n职称: " << title << "\n系别: " << department << "\n基本工资: " << salary << "\n指导的学生: \n";

auto it = teacherStudentMap.find(const\_cast<Teacher\*>(this));

if (it != teacherStudentMap.end()) {

for (const auto& student : it->second) {

std::cout << "\t" << student->getName() << "\n";

}

}

else

std::cout << "\t" << "无" << "\n";

//std::cout << std::endl;

}

double Teacher::getSalary() const {

return salary;

}

void Teacher::setSalary(double salary) {

this->salary = salary;

}

std::ostream& operator<<(std::ostream& os, const Teacher& t) {

t.outData(std::map<Teacher\*, std::vector<Student\*>>());

return os;

}

OrdinaryTeacher.h

#ifndef ORDINARYTEACHER\_H

#define ORDINARYTEACHER\_H

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

#include <map>

//#include "Student.h"

#include "Teacher.h"

//#include "Person.h"

//class Teacher;

class Person; // 前向声明

class student;

class OrdinaryTeacher : public Teacher {

private:

double hourlyWage;

int teachingHours;

public:

void inputData() override;

void outData(const std::map<Teacher\*, std::vector<Student\*>>& teacherStudentMap) const override;

double getIncome() const override;

};

#endif

OrdinaryTeacher.cpp

#include "OrdinaryTeacher.h"

void OrdinaryTeacher::inputData() {

Teacher::inputData();

std::cout << "输入课时费: ";

std::cin >> hourlyWage;

std::cout << "输入教学时长: ";

std::cin >> teachingHours;

}

void OrdinaryTeacher::outData(const std::map<Teacher\*, std::vector<Student\*>>& teacherStudentMap) const {

Teacher::outData(teacherStudentMap);

std::cout << "课时费: " << hourlyWage << "\n教学时长: " << teachingHours << "\n总收入: " << getIncome() << std::endl;

}

double OrdinaryTeacher::getIncome() const {

return salary + hourlyWage \* teachingHours;

}

Dean.h

#ifndef DEAN\_H

#define DEAN\_H

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

#include <map>

//#include "Student.h"

#include "Teacher.h"

//#include "Person.h"

//class Teacher;

class Person; // 前向声明

class student;

class Dean : public Teacher {

private:

double performancePay;

public:

void inputData() override;

void outData(const std::map<Teacher\*, std::vector<Student\*>>& teacherStudentMap) const override;

double getIncome() const override;

};

#endif

Dean.cpp

#include "Dean.h"

void Dean::inputData() {

Teacher::inputData();

std::cout << "输入绩效: ";

std::cin >> performancePay;

}

void Dean::outData(const std::map<Teacher\*, std::vector<Student\*>>& teacherStudentMap) const {

Teacher::outData(teacherStudentMap);

std::cout << "绩效: " << performancePay << "\n总收入: " << getIncome() << std::endl;

}

double Dean::getIncome() const {

return salary + performancePay;

}

President.h

#ifndef PRESIDENT\_H

#define PRESIDENT\_H

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

#include <map>

//#include "Student.h"

#include "Teacher.h"

//#include "Person.h"

//class Teacher;

class Person; // 前向声明

class student;

class President : public Teacher {

private:

double annualSalary;

public:

void inputData() override;

void outData(const std::map<Teacher\*, std::vector<Student\*>>& teacherStudentMap) const override;

double getIncome() const override;

};

#endif

President.cpp

#include "President.h"

void President::inputData() {

Teacher::inputData();

std::cout << "输入年薪: ";

std::cin >> annualSalary;

}

void President::outData(const std::map<Teacher\*, std::vector<Student\*>>& teacherStudentMap) const {

Teacher::outData(teacherStudentMap);

std::cout << "年薪: " << annualSalary << "\n总收入: " << getIncome() << std::endl;

}

double President::getIncome() const {

return salary + annualSalary / 12;

}

Main.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

#include <map>

#include "Person.h"

#include "Student.h"

#include "Teacher.h"

#include "OrdinaryTeacher.h"

#include"Dean.h"

#include "President.h"

//class Teacher; // 前向声明

//class Student;

void displayMenu() { // 显示操作菜单

std::cout << "\n=========通过数字选择操作:=========" << std::endl;

std::cout << "1. 添加人员 (学生/教师)" << std::endl;

std::cout << "2. 显示所有成员" << std::endl;

std::cout << "3. 通过姓名查询人员" << std::endl;

std::cout << "4. 通过姓名删除人员" << std::endl;

std::cout << "5. 管理学生的导师" << std::endl;

std::cout << "6. 管理导师的学生" << std::endl;

std::cout << "7. 同类对象按年龄排序" << std::endl;

std::cout << "8. 退出" << std::endl;

std::cout << "===================================\n" << std::endl;

}

void addPerson(std::vector<Person\*>& persons, std::map<Teacher\*, std::vector<Student\*>>& teacherStudentMap) { // 1 号操作菜单

int choice;

std::cout << "\n选择添加人员的种类:" << std::endl;

std::cout << "1. 普通教师" << std::endl;

std::cout << "2. 系主任" << std::endl;

std::cout << "3. 院长" << std::endl;

std::cout << "4. 学生\n" << std::endl;

std::cin >> choice;

Person\* person = nullptr;

Teacher\* teacher = nullptr;

switch (choice) {

case 1:

teacher = new OrdinaryTeacher();

teacher->inputData();

person = teacher;

break;

case 2:

teacher = new Dean();

teacher->inputData();

person = teacher;

break;

case 3:

teacher = new President();

teacher->inputData();

person = teacher;

break;

case 4:

person = new Student();

person->inputData();

break;

default:

std::cout << "\n非法选项！" << std::endl;

return;

}

if (person) {

persons.push\_back(person);

if (auto\* student = dynamic\_cast<Student\*>(person)) {

Teacher\* mentor = nullptr;

std::string mentorName;

std::cout << "输入学生导师姓名: ";

std::cin >> mentorName;

for (auto\* p : persons) {

auto\* t = dynamic\_cast<Teacher\*>(p);

if ( t && t->getName() == mentorName) {

mentor = t;

break;

}

}

if (mentor) {

student->setMentor(mentor);

teacherStudentMap[mentor].push\_back(student);

}

else {

std::cout << "\n没有找到该导师!" << std::endl;

}

}

}

}

void displayAllPersons(const std::vector<Person\*>& persons, const std::map<Teacher\*, std::vector<Student\*>>& teacherStudentMap) {

for (const auto& person : persons) {

//std::cout << \*person << std::endl;

person->outData(teacherStudentMap);

}

}

void queryPersonByName(const std::vector<Person\*>& persons, const std::map<Teacher\*, std::vector<Student\*>>& teacherStudentMap) {

std::string name;

std::cout << "\n输入需要查询的人员姓名: ";

std::cin >> name;

for (const auto& person : persons) {

if (person->getName() == name) {

//std::cout << \*person << std::endl;

person->outData(teacherStudentMap);

return;

}

}

std::cout << "\n没有找到该人员!" << std::endl;

}

void deletePersonByName(std::vector<Person\*>& persons, std::map<Teacher\*, std::vector<Student\*>>& teacherStudentMap) {

std::string name;

std::cout << "\n输入需要删除的人员姓名: ";

std::cin >> name;

for (auto it = persons.begin(); it != persons.end(); ++it) {

if ((\*it)->getName() == name) {

if (auto\* teacher = dynamic\_cast<Teacher\*>(\*it)) {

auto& students = teacherStudentMap[teacher];

for (auto\* student : students) {

student->setMentor(nullptr);

}

teacherStudentMap.erase(teacher);

}

else if (auto\* student = dynamic\_cast<Student\*>(\*it)) {

Teacher\* mentor = student->getMentor();

if (mentor) {

auto& students = teacherStudentMap[mentor];

students.erase(std::remove(students.begin(), students.end(), student), students.end());

}

}

delete\* it;

persons.erase(it);

std::cout << "\n人员已删除." << std::endl;

return;

}

}

std::cout << "\n没有找到该人员!" << std::endl;

}

void manageStudentMentor(std::vector<Person\*>& persons, std::map<Teacher\*, std::vector<Student\*>>& teacherStudentMap) {

std::string studentName;

std::cout << "\n输入学生姓名: ";

std::cin >> studentName;

Student\* student = nullptr;

for (auto\* person : persons) {

auto\* s = dynamic\_cast<Student\*>(person);

if ( s && s->getName() == studentName) {

student = s;

break;

}

}

if (!student) {

std::cout << "\n没有找到该学生!" << std::endl;

return;

}

std::cout << "\n1. 添加导师" << std::endl;

std::cout << "2. 变更导师" << std::endl;

std::cout << "3. 移除导师\n" << std::endl;

int choice;

std::cin >> choice;

std::string mentorName;

Teacher\* mentor = nullptr;

switch (choice) {

case 1:

if (student->getMentor()) {

std::cout << "\n学生已有导师，不允许继续添加！请使用变更导师选项." << std::endl; // 按题目要求一个学生只能有一个指导老师

break;

}

case 2:

std::cout << "\n输入导师姓名: ";

std::cin >> mentorName;

for (auto\* person : persons) {

auto\* t = dynamic\_cast<Teacher\*>(person);

if ( t && t->getName() == mentorName) {

mentor = t;

break;

}

}

if (!mentor) {

std::cout << "\n没有找到该导师!" << std::endl;

return;

}

if (choice == 2 && student->getMentor()) {

auto& oldStudents = teacherStudentMap[student->getMentor()];

oldStudents.erase(std::remove(oldStudents.begin(), oldStudents.end(), student), oldStudents.end());

}

student->setMentor(mentor);

teacherStudentMap[mentor].push\_back(student);

std::cout << "\n导师 " << (choice == 1 ? "已添加." : "已修改.") << std::endl;

break;

case 3:

if (auto\* currentMentor = student->getMentor()) {

auto& students = teacherStudentMap[currentMentor];

students.erase(std::remove(students.begin(), students.end(), student), students.end());

student->setMentor(nullptr);

std::cout << "\n导师已移除." << std::endl;

}

else {

std::cout << "\n该学生没有导师." << std::endl;

}

break;

default:

std::cout << "\n非法选项!" << std::endl;

break;

}

}

void manageTeachersStudents(std::vector<Person\*>& persons, std::map<Teacher\*, std::vector<Student\*>>& teacherStudentMap) {

std::string teacherName;

std::cout << "\n输入导师姓名: ";

std::cin >> teacherName;

Teacher\* teacher = nullptr;

for (auto\* person : persons) {

auto\* t = dynamic\_cast<Teacher\*>(person);

if ( t && t->getName() == teacherName) {

teacher = t;

break;

}

}

if (!teacher) {

std::cout << "\n没有找到该导师!" << std::endl;

return;

}

std::cout << "\n1. 添加学生" << std::endl;

std::cout << "2. 移除学生\n" << std::endl;

int choice;

std::cin >> choice;

std::string studentName;

Student\* student = nullptr;

switch (choice) {

case 1:

std::cout << "\n输入学生姓名: ";

std::cin >> studentName;

for (auto\* person : persons) {

auto\* s = dynamic\_cast<Student\*>(person);

if ( s && s->getName() == studentName) {

student = s;

break;

}

}

if (!student) {

std::cout << "\n没有找到该学生!" << std::endl;

return;

}

if (student->getMentor()) { // 题目要求一个学生只能一个导师，如果学生已有导师，就更新导师信息

auto& oldStudents = teacherStudentMap[student->getMentor()];

std::cout << "\n学生已有导师，已将该学生导师信息更新." << std::endl;

oldStudents.erase(std::remove(oldStudents.begin(), oldStudents.end(), student), oldStudents.end());

}

student->setMentor(teacher);

teacherStudentMap[teacher].push\_back(student);

std::cout << "\n学生已添加." << std::endl;

break;

case 2:

std::cout << "\n输入学生姓名: ";

std::cin >> studentName;

for (auto\* person : persons) {

auto\* s = dynamic\_cast<Student\*>(person);

if ( s && s->getName() == studentName) {

student = s;

break;

}

}

if (!student) {

std::cout << "\n没有找到该学生!" << std::endl;

return;

}

if (student->getMentor() == teacher) {

auto& students = teacherStudentMap[teacher];

students.erase(std::remove(students.begin(), students.end(), student), students.end());

student->setMentor(nullptr);

std::cout << "\n学生已移除." << std::endl;

}

else {

std::cout << "\n这位老师不是该学生的导师." << std::endl;

}

break;

default:

std::cout << "\n非法选项!" << std::endl;

break;

}

}

void classifyAndSortPersonsByAge(const std::vector<Person\*>& persons, std::map<Teacher\*, std::vector<Student\*>>& teacherStudentMap) {

std::vector<Student\*> students;

std::vector<OrdinaryTeacher\*> ordinaryTeachers;

std::vector<Dean\*> deans;

std::vector<President\*> presidents;

for (auto person : persons) {

if (auto\* student = dynamic\_cast<Student\*>(person)) {

students.push\_back(student);

}

else if (auto\* ordinaryTeacher = dynamic\_cast<OrdinaryTeacher\*>(person)) {

ordinaryTeachers.push\_back(ordinaryTeacher);

}

else if (auto\* dean = dynamic\_cast<Dean\*>(person)) {

deans.push\_back(dean);

}

else if (auto\* president = dynamic\_cast<President\*>(person)) {

presidents.push\_back(president);

}

}

// 将不同类按年龄排序

auto sortByAge = [](auto& container) {

std::sort(container.begin(), container.end(), [](auto\* a, auto\* b) {

return a->getAge() < b->getAge();

});

};

sortByAge(students);

sortByAge(ordinaryTeachers);

sortByAge(deans);

sortByAge(presidents);

// 输出排序结果

std::cout << "\n学生按年龄排序:" << std::endl;

for (const auto& student : students) {

student->outData(teacherStudentMap);

}

std::cout << "\n普通教师按年龄排序:" << std::endl;

for (const auto& ordinaryTeacher : ordinaryTeachers) {

ordinaryTeacher->outData(teacherStudentMap);

}

std::cout << "\n系主任按年龄排序:" << std::endl;

for (const auto& dean : deans) {

dean->outData(teacherStudentMap);

}

std::cout << "\n院长按年龄排序:" << std::endl;

for (const auto& president : presidents) {

president->outData(teacherStudentMap);

}

}

int main() {

std::vector<Person\*> persons;

std::map<Teacher\*, std::vector<Student\*>> teacherStudentMap;

bool exitProgram = false;

while (!exitProgram) {

displayMenu(); // 显示操作菜单

int choice;

std::cin >> choice;

switch (choice) {

case 1:

addPerson(persons, teacherStudentMap);

break;

case 2:

displayAllPersons(persons, teacherStudentMap);

break;

case 3:

queryPersonByName(persons, teacherStudentMap);

break;

case 4:

deletePersonByName(persons, teacherStudentMap);

break;

case 5:

manageStudentMentor(persons, teacherStudentMap);

break;

case 6:

manageTeachersStudents(persons, teacherStudentMap);

break;

case 7:

classifyAndSortPersonsByAge(persons, teacherStudentMap);

break;

case 8:

exitProgram = true;

break;

default:

std::cout << "\n非法选项!" << std::endl;

break;

}

}

// 清理内存

for (const auto& person : persons) {

delete person;

}

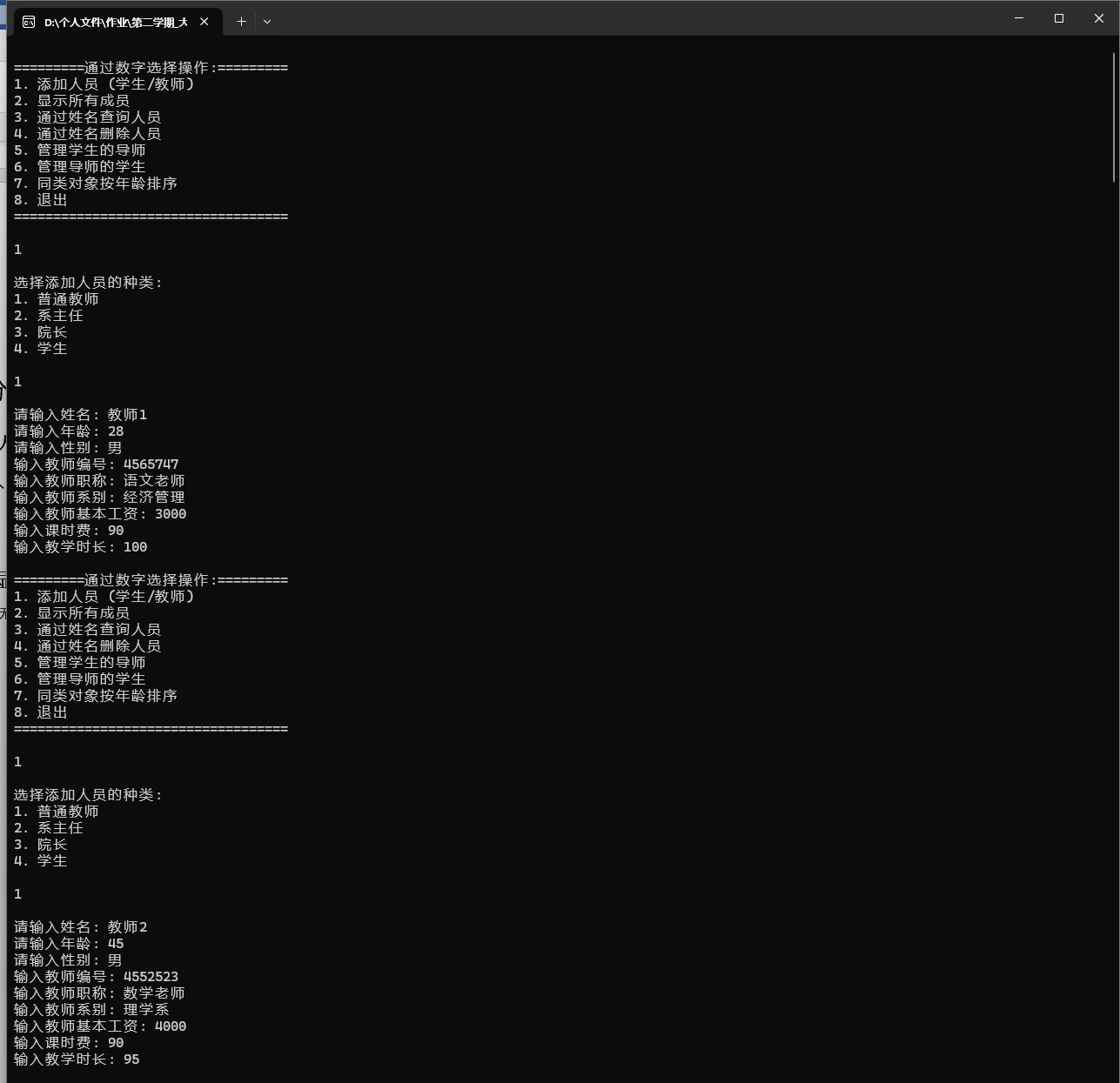
return 0;

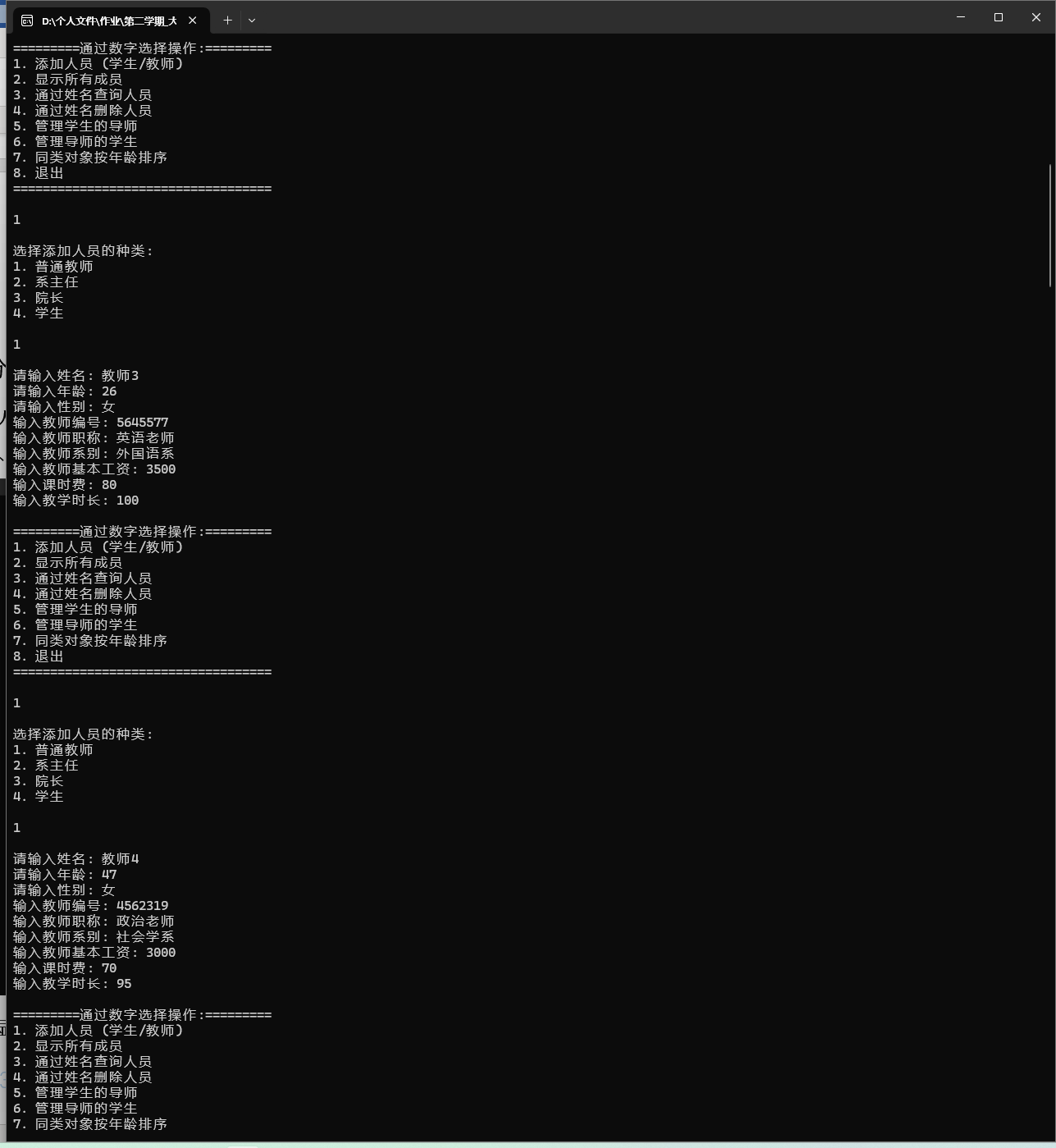
}

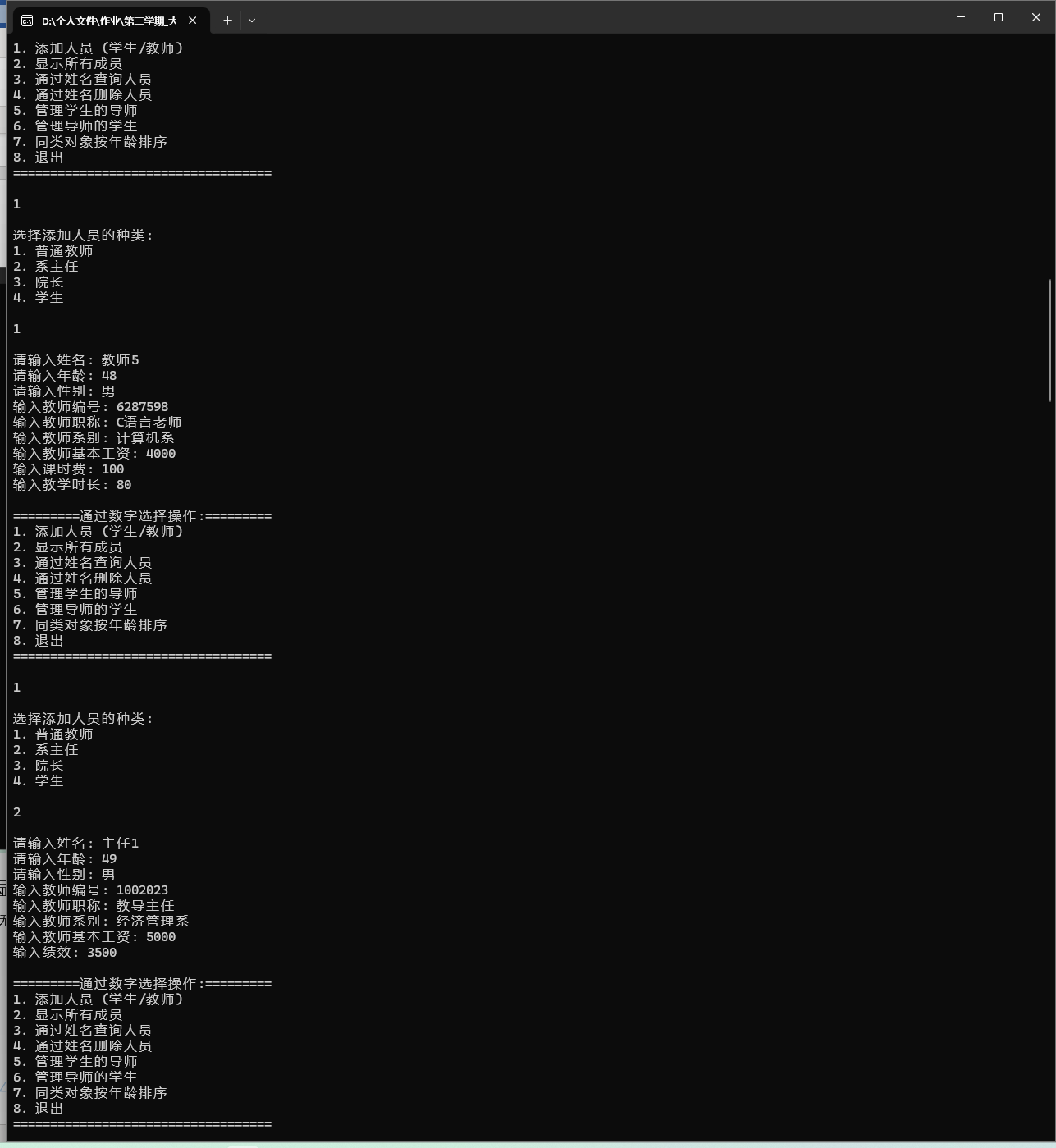
五、实验结果（10分）

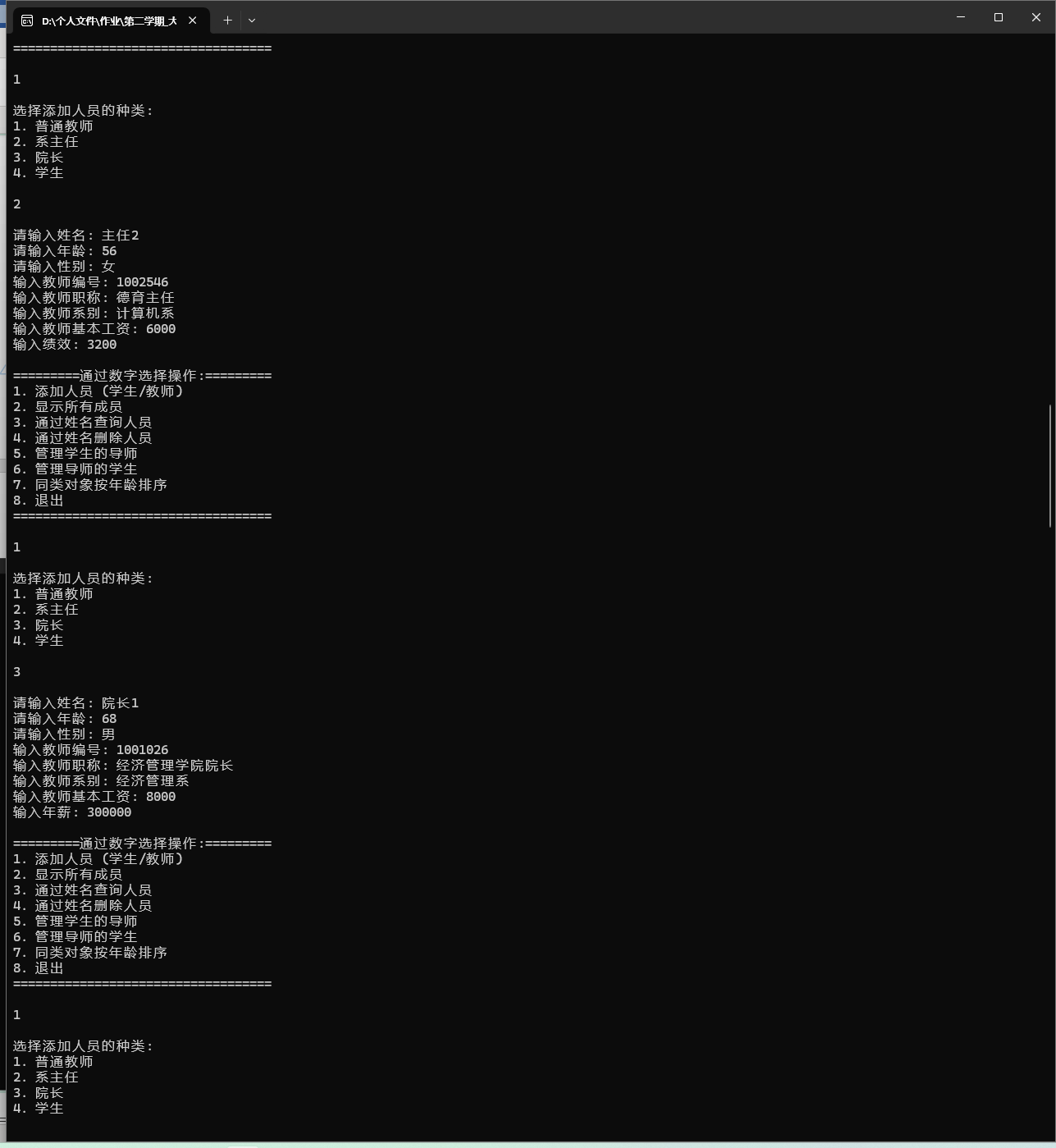
功能测试1.添加人员 (学生/教师)

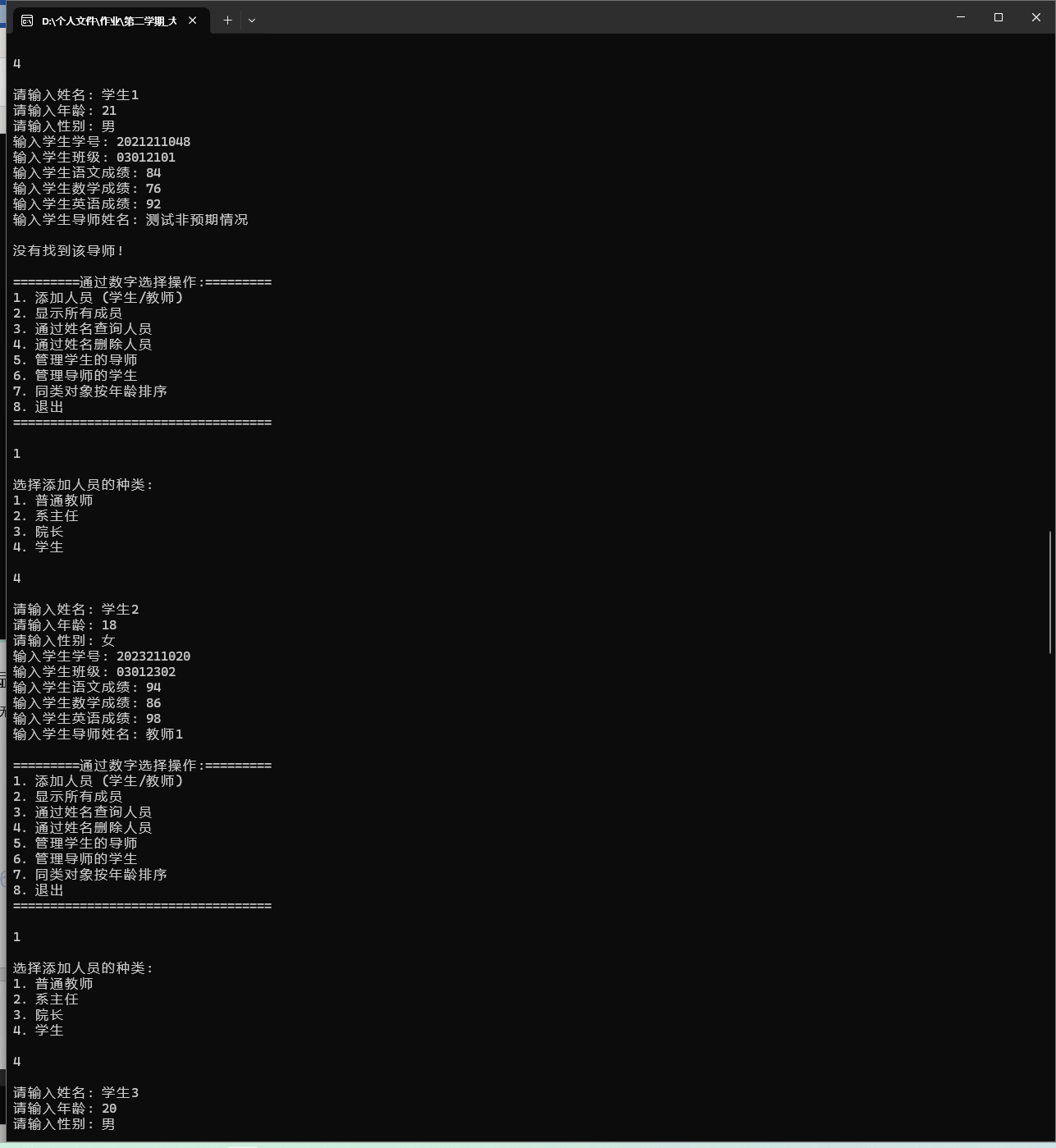
在录入学生1时测试了输入预期外的导师姓名处理情况，结果正常。

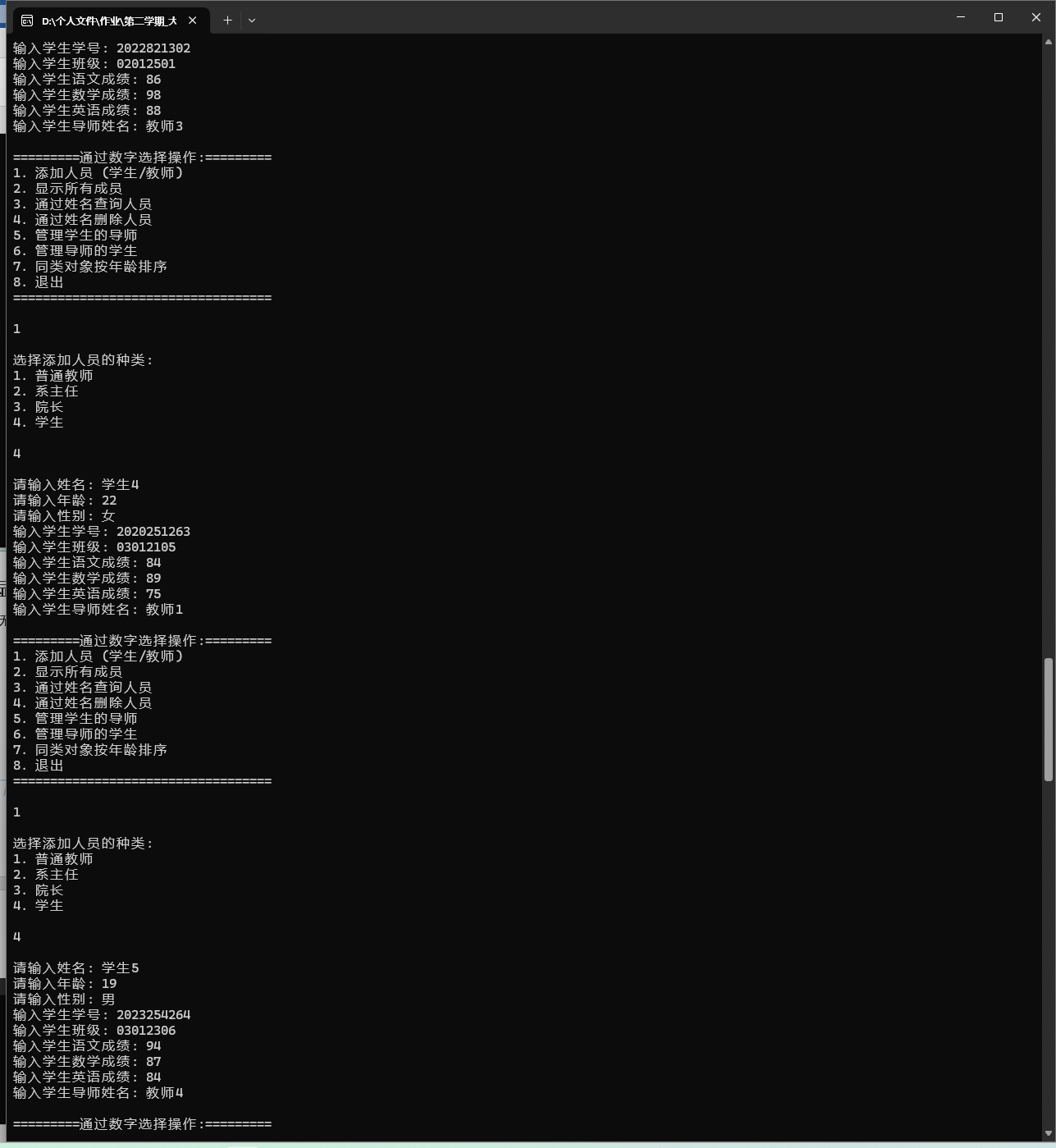






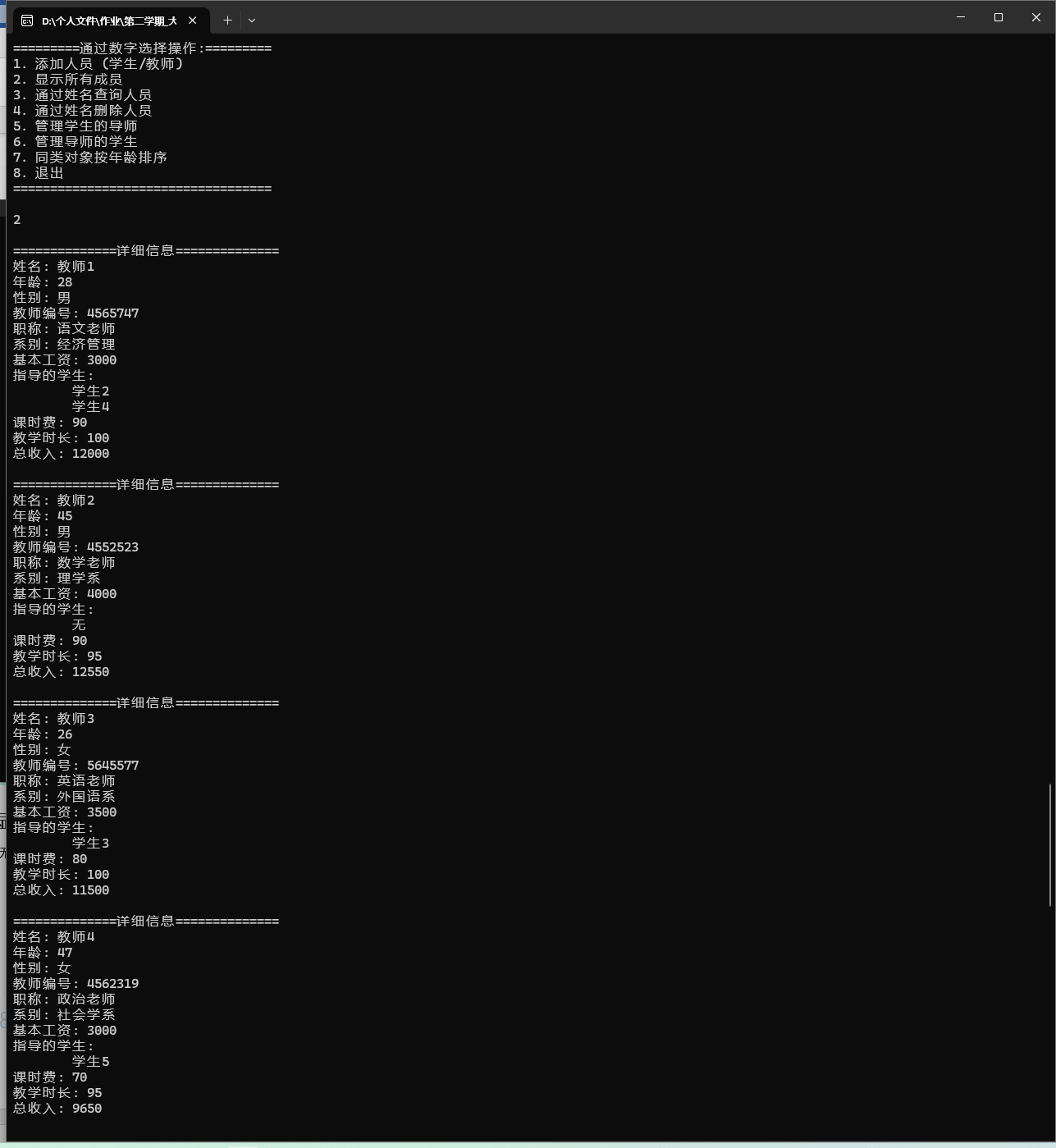


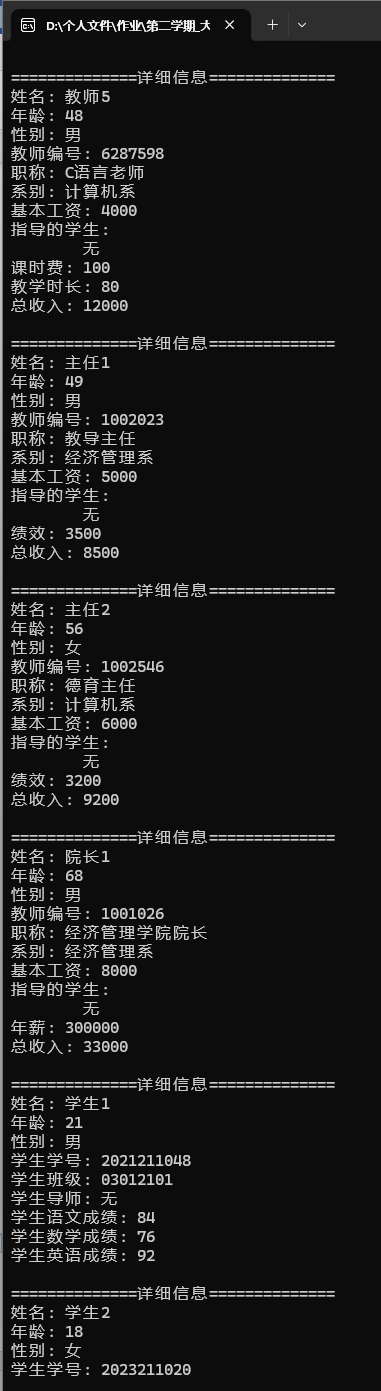


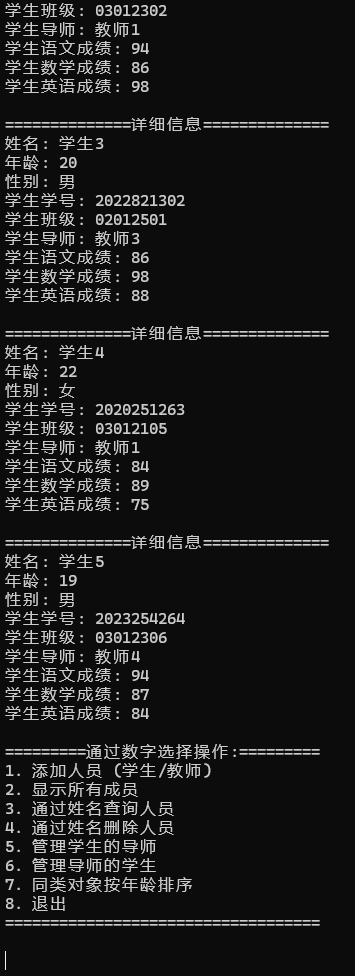


功能测试2. 显示所有成员

在输出时，老师无指导的学生输出“无”，学生无指导老师输出“无”，功能正常。

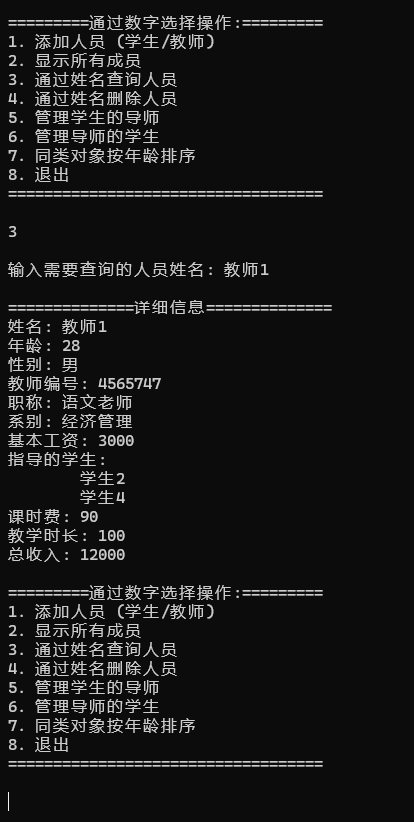




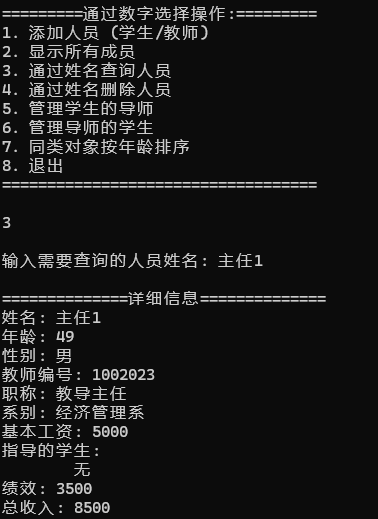


功能测试3. 通过姓名查询人员

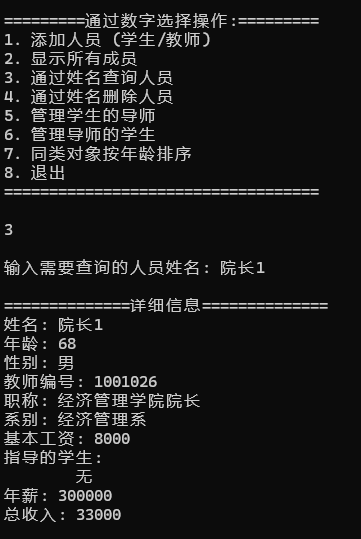
查询教师（可输出指导的学生）：



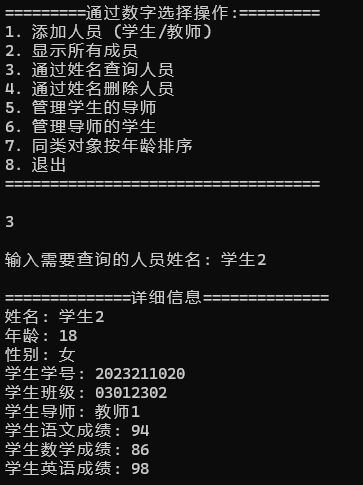
查询系主任：



查询院长：

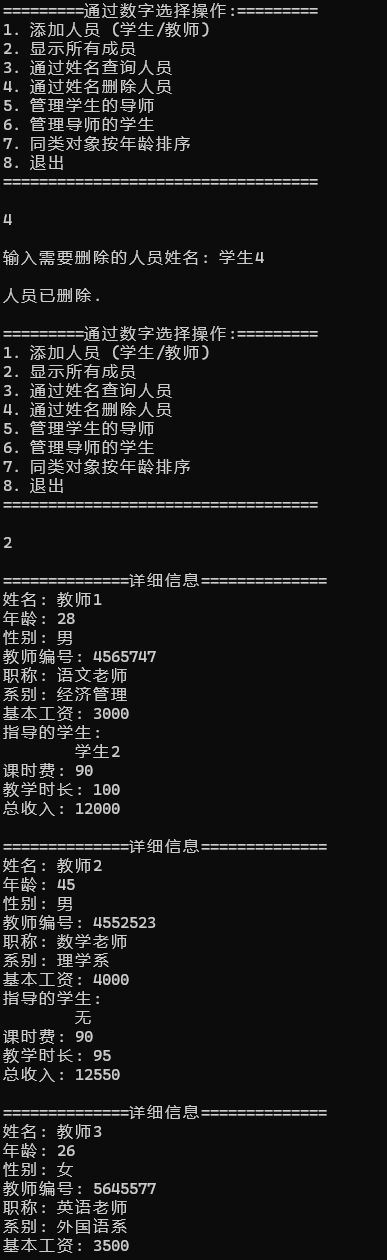


查询学生：

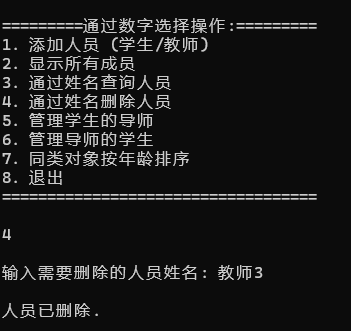


功能测试4. 通过姓名删除人员

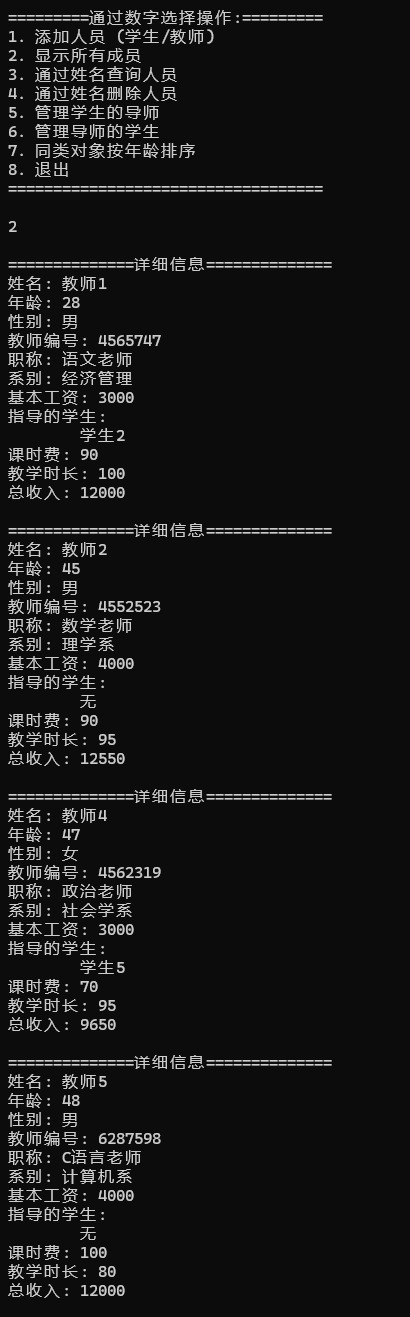
删除学生（同时会删除他的老师指导该学生的信息，保持一致性。）

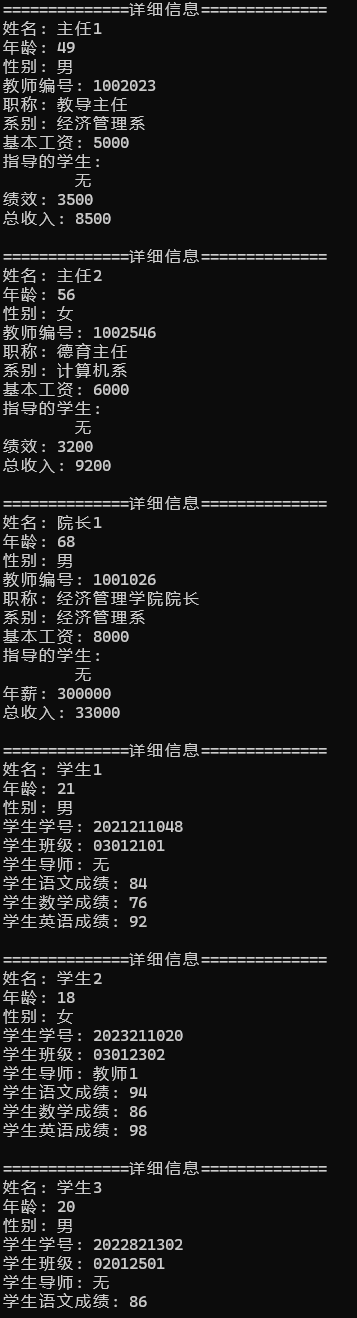


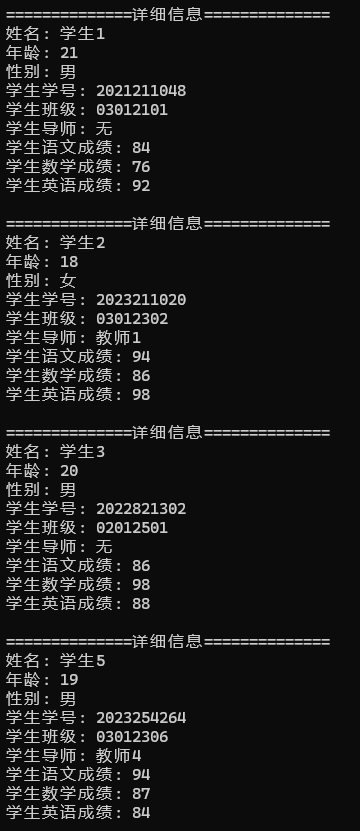
删除老师（会同时删除所有他指导的学生的导师信息）



输出所有成员信息验证（注意输出的教师1信息，指导的学生中移除了被删除的学生4。注意输出的学生3信息 ，指导老师移除了被删除的教师3）

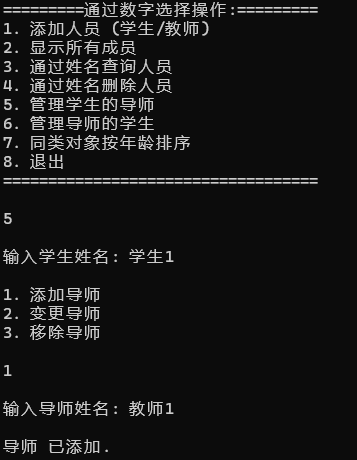




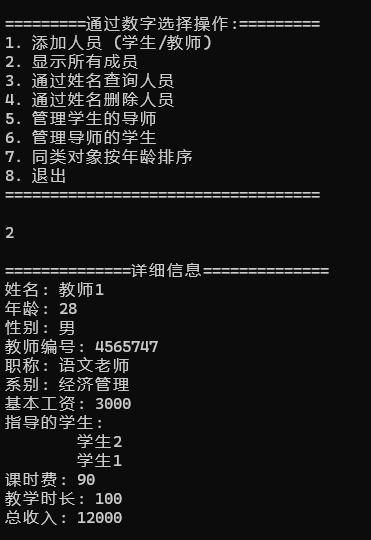


功能测试5. 管理学生的导师

测试添加导师

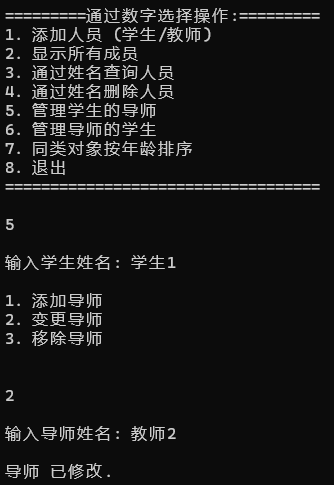


输出验证添加成功（学生和老师的信息都更新了）

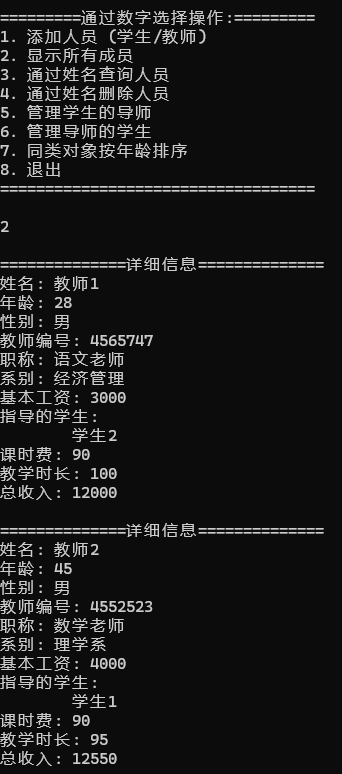


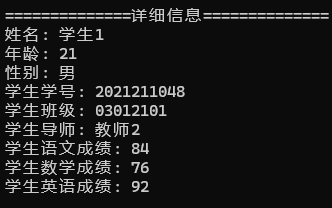


测试变更导师

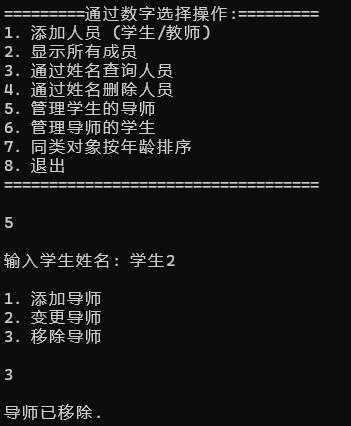


输出验证更改成功(原老师、新老师和学生的信息都更新了)

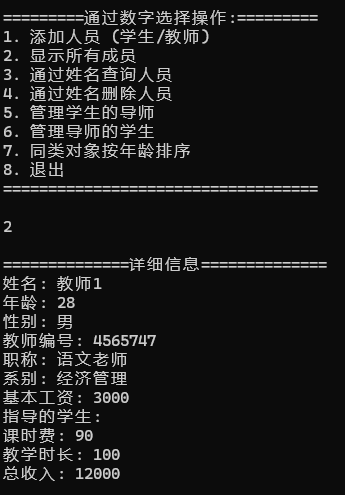


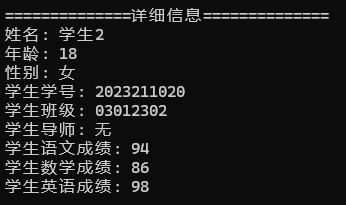


测试移除导师

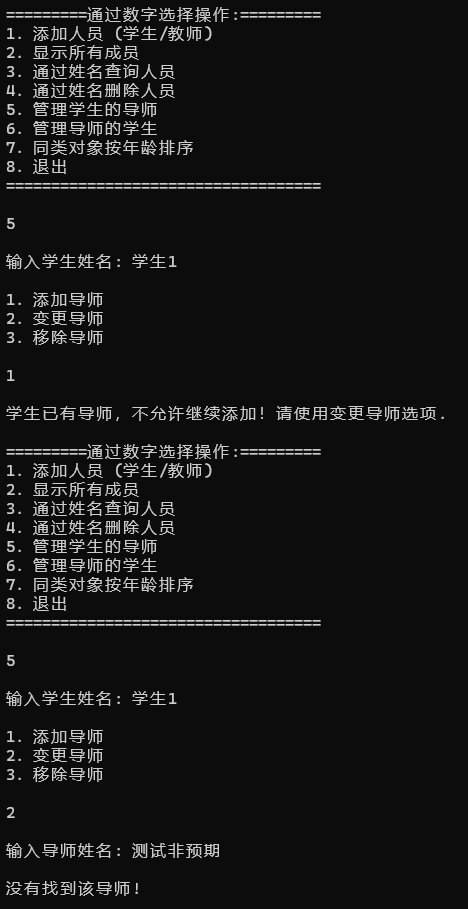


输出验证移除成功（学生和老师的信息都更新了）



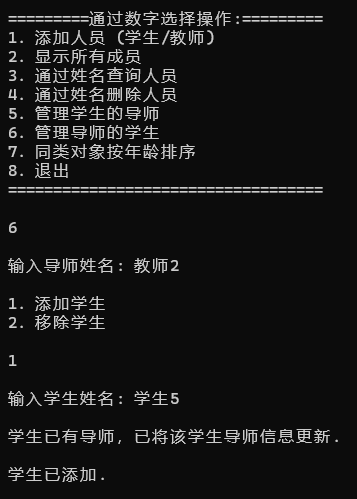


测试非预期情况处理

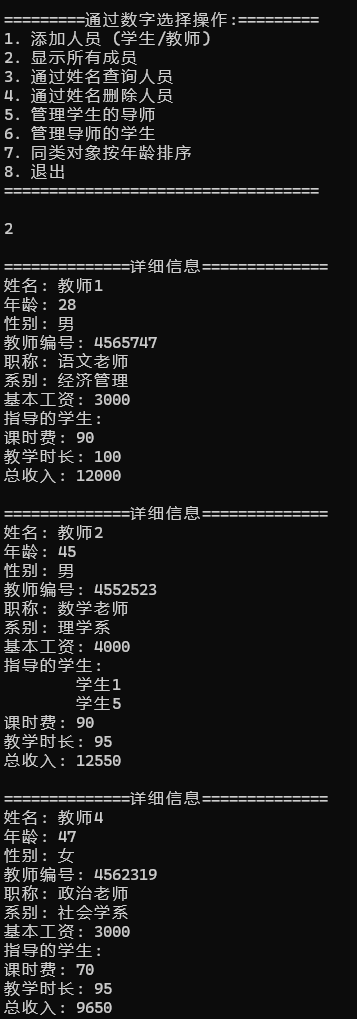


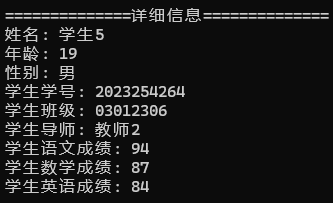
功能测试6. 管理导师的学生(添加已有导师的学生会直接变更他的导师信息)

测试添加学生

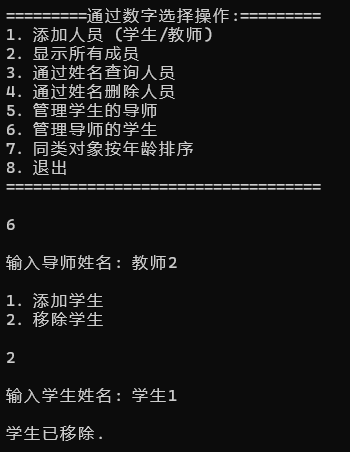


输出验证添加成功（原导师指导的信息已变更）

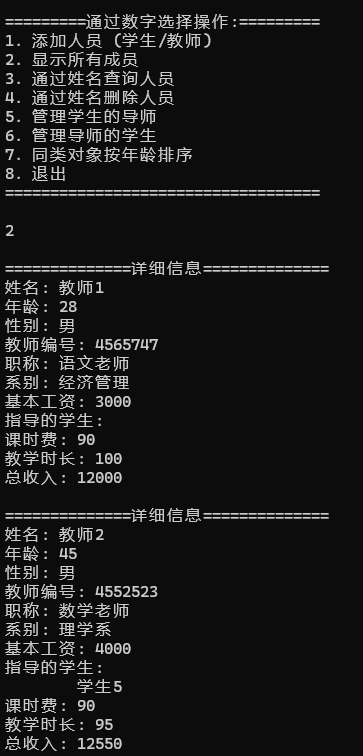




测试移除学生

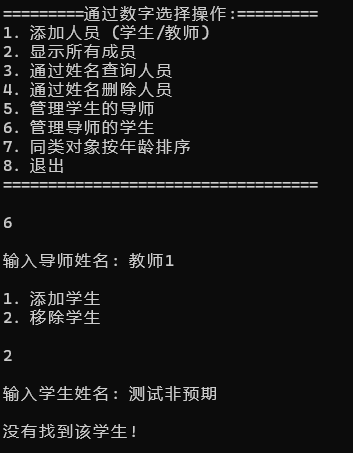


输出验证移除成功（学生和老师的信息都更新了）

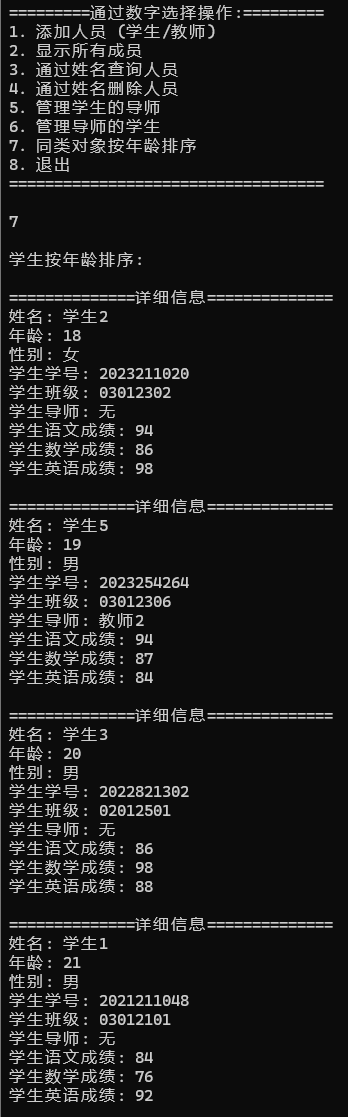


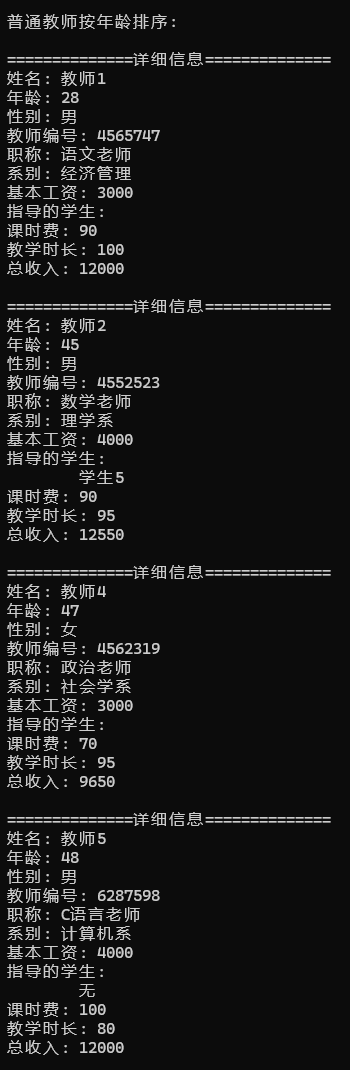


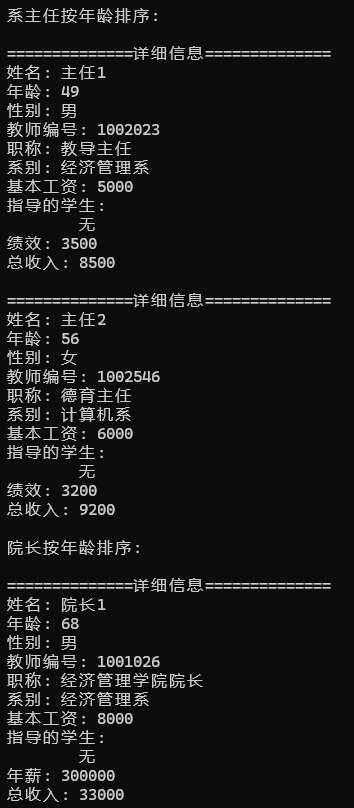
测试非预期情况处理



功能测试7. 同类对象按年龄排序







功能测试8. 退出

