

# 课 程 设 计 报 告

课程名称       计算机程序设计基础 2      

班	级	<u>      无 26      </u>
学	号	<u>      2022010650      </u>
姓	名	<u>      潘徐成      </u>

2023 年 6 月 27 日

# 目录

---

该作业已托管至 [GitHub](#)。

# 一、设计内容与设计要求

---

## 1. 课程设计目的

面向对象程序设计课程设计是集中实践性环节之一，是学习完《计算机程序设计基础2》C++面向对象程序设计课程后进行的一次全面的综合练习。要求学生达到熟练掌握C++语言的基本知识和技能；基本掌握面向对象程序设计的思想和方法；能够利用所学的基本知识和技能，解决简单的面向对象程序设计问题，从而提高动手编程解决实际问题的能力。尤其重视创新思维培养。

## 2. 课题题目

- 1) 学生成绩管理系统（或公司人事管理系统）

## 3. 文档设计要求

3.1 设计课题题目：每个同学都单独完成1道课题。后面有范题，仅供同学们参考，不列入本次课程设计的课题。

3.2 对于程设题目，按照范题的格式。自行虚构软件需求。并按照第4点要求，编写设计文档。基本要求系统中设计的类的数目不少于4个，每个类中要有各自的属性（多于3个）和方法（多于3个）；需要定义一个抽象类，采用继承方式派生这些类。并设计一个多重继承的派生类。在程序设计中，引入虚函数的多态性、运算符重载等机制。

## 4. 程序设计的基本要求

- （1）要求利用面向对象的方法以及C++的编程思想来完成系统的设计；
- （2）要求在设计的过程中，建立清晰的类层次；
- （3）根据课题完成以下主要工作：①完成系统需求分析：包括系统设计目的与意义；系统功能需求（系统流程图）；输入输出的要求。②完成系统总体设计：包括系统功能分析；系统功能模块划分与设计（系统功能模块图）。③完成系统详细设计：数据文件；类层次图；界面设计与各功能模块实现。④系统调试：调试出现的主要问题，编译语法错误及修改，重点是运行逻辑问题修改和调整。⑤使用说明书及编程体会：说明如何使用你编写的程序，详细列出每一步的操作步骤。⑥关键源程序（带注释）。
- （4）自己设计测试数据，将测试数据存在文件中，通过文件进行数据读写来获得测试结果。
- （5）按规定格式完成课程设计报告，并在网络学堂上按时提交。
- （6）不得抄袭他人程序、课程设计报告，每个人应独立完成，在程序和设计报告中体现自己的个性设计。

## 5. 进度安排

小学期第2周	对学生成绩管理系统的总体架构进行规划	设计学生成绩管理系统的核心类	完成链表、学生类、课程类的编写
	设计数据库类及各项功能	完成添加、保存、读取学生成绩信息功能的编写	完成修改、删除学生成绩信息功能的编写
	设计用户界面类及各项功能	完成查询学生成绩信息功能的编写、调整输出格式	完成输出、打印学生成绩信息功能的编写
	设计账号类及各项功能	完成账号等各项功能的编写	为账号进一步设计、优化用户界面
	在不同编程环境配置中测试程序的兼容性	进行运行测试，观察程序在极端条件下的运行情况	调试程序，寻找漏洞与错误

## 二、系统需求分析

---

学生成绩管理系统记录了学生的成绩情况，它包括：学生学号、学生姓名、课程名称、课程类型（必选、限选及任选）、课程学分、课程成绩、课程绩点等。试设计学生成绩管理系统，使之能提供以下功能：

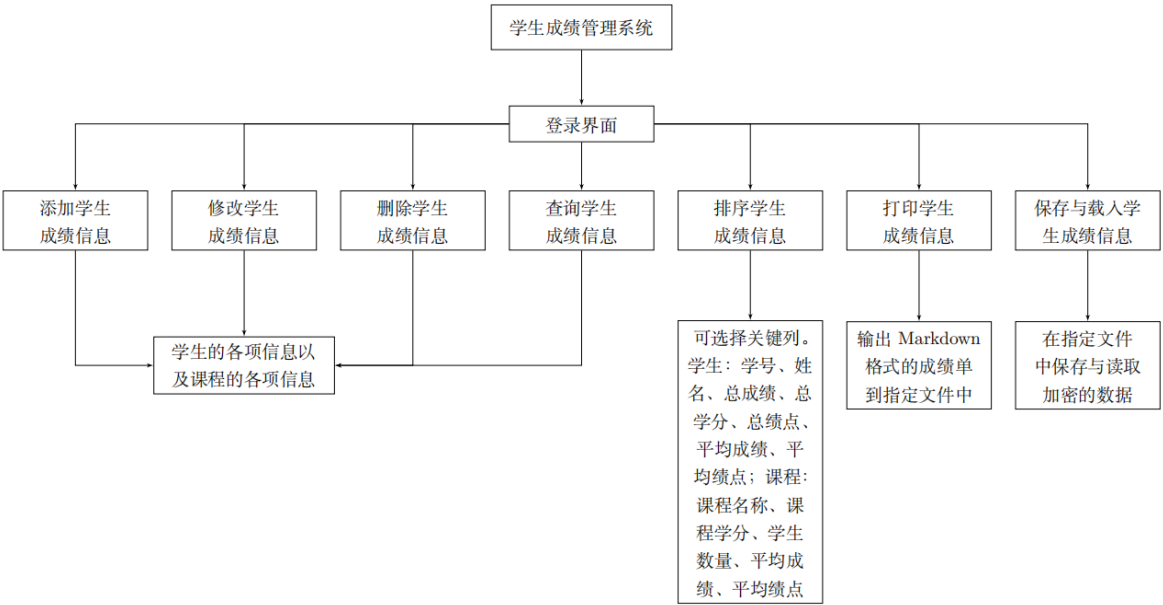
- 录入学生的信息及成绩。通过程序中的提示从键盘输入数据。
- 修改学生的信息及成绩。可以对学生的学号、姓名以及所选科目的任何信息进行修改，并实现对应课程信息的自动更新。
- 删除学生的信息及成绩。可以对任何学生和课程进行删除。
- 查询学生的信息及成绩。可以通过学号和姓名查询学生信息，通过课程名称查询课程信息。
- 排序学生的信息及成绩。可以以学号、姓名、总成绩、总学分、总绩点、平均成绩、平均绩点等排序学生，以课程名称、课程学分、学生数量、平均成绩、平均绩点等排序课程。
- 打印学生的信息及成绩。可以将学生信息及课程信息输出到 `Markdown` 文件。
- 保存与载入学生的信息及成绩。可以读取和载入学生及课程信息。
- 以菜单方式工作，便于用户选择。

### 三、总体设计

学生成绩管理系统包含七个大的功能，分别是：录入学生成绩信息、修改学生成绩信息、删除学生成绩信息、查询学生成绩信息、排序学生成绩信息、打印学生成绩信息、保存与载入学生信息。学生的信息主要包含学号、姓名、选课信息、总学分、平均成绩、平均绩点等。课程的信息主要包含课程名称、课程学分、课程类型、选课学生信息、平均成绩、平均绩点等。

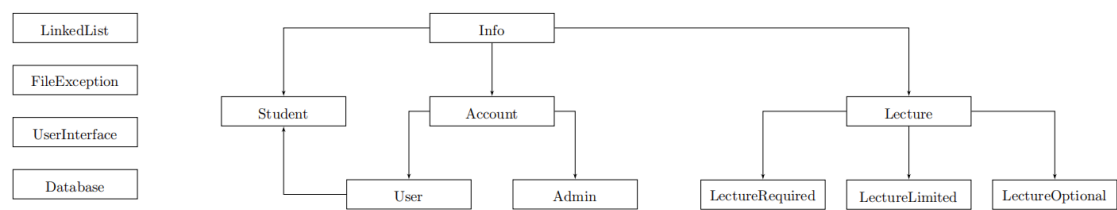
- 在录入学生成绩信息时根据系统提示逐一输入。每输入完一条信息，系统会提示是否继续输入，用户可以选择继续输入或返回上一级。
- 在修改学生成绩信息时，用户首先输入要修改的学生姓名或学号，系统会进行检索。如果系统中有该学生的相关信息，则系统则会提示用户进行修改。如果系统中没有该学生的相关信息，则系统会给相关提示，并让用户返回主菜单。同时，用户可以输入要修改的课程名称，修改课程信息。
- 在删除学生成绩信息时，用户首先输入要修改的学生姓名或学号，系统会进行检索。如果系统中有该学生的相关信息，则系统则会提示用户进行删除。如果系统中没有该学生的相关信息，则系统会给相关提示，并让用户返回主菜单。同时，用户可以输入要删除的课程名称，删除课程信息。
- 在查询学生成绩信息时，用户首先输入要查询的学生姓名或学号，系统会进行检索。如果系统中有该学生的相关信息，则系统则会展示用户查询的学生信息。如果系统中没有该学生的相关信息，则系统会给相关提示，并让用户返回主菜单。同时，用户可以输入要查询的课程名称，查询课程信息。
- 在排序学生课程信息时，用户可以通过选择菜单选择进行排序的关键列，并且选择升序和降序。学生的关键列有：学号、姓名、总成绩、总学分、总绩点、平均成绩、平均绩点；课程的关键列有：课程名称、课程学分、学生数量、平均成绩、平均绩点。
- 在打印学生成绩信息时，用户首先输入要打印的学生姓名或学号，系统会进行检索。如果系统中有该学生的相关信息，则系统则会以输出到 `output_student_学号_姓名.md` 的方式打印学生成绩单。用户也可以选择打印全部学生信息（输出到 `output_student_all.md`）、打印课程信息（输出到 `output_lecture_课程名称.md`）、打印全部课程信息（输出到 `output_lecture_all.md`）。

学生考勤管理系统中功能模块图：

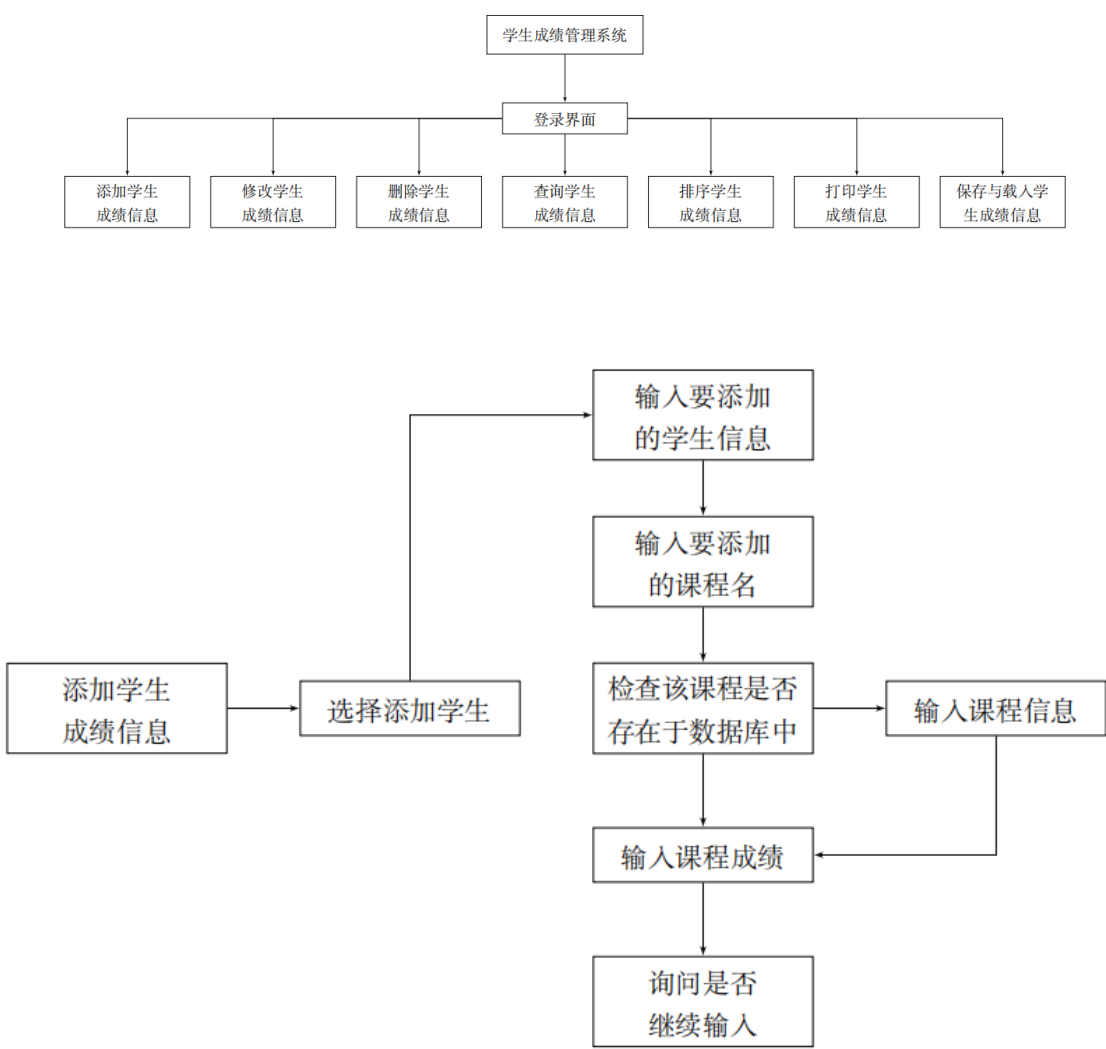


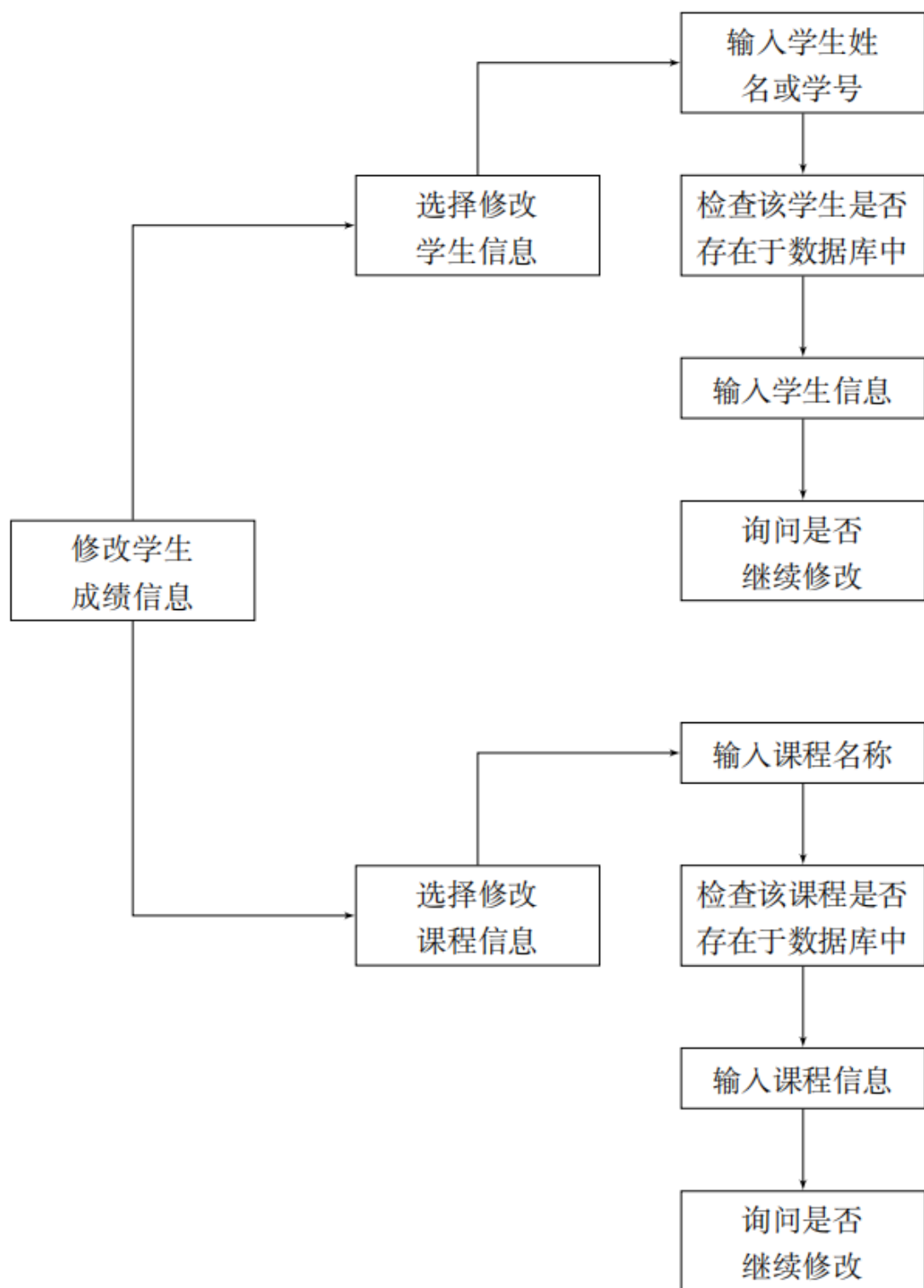
# 四、详细设计

## 1. 学生成绩管理系统中类的类层次图

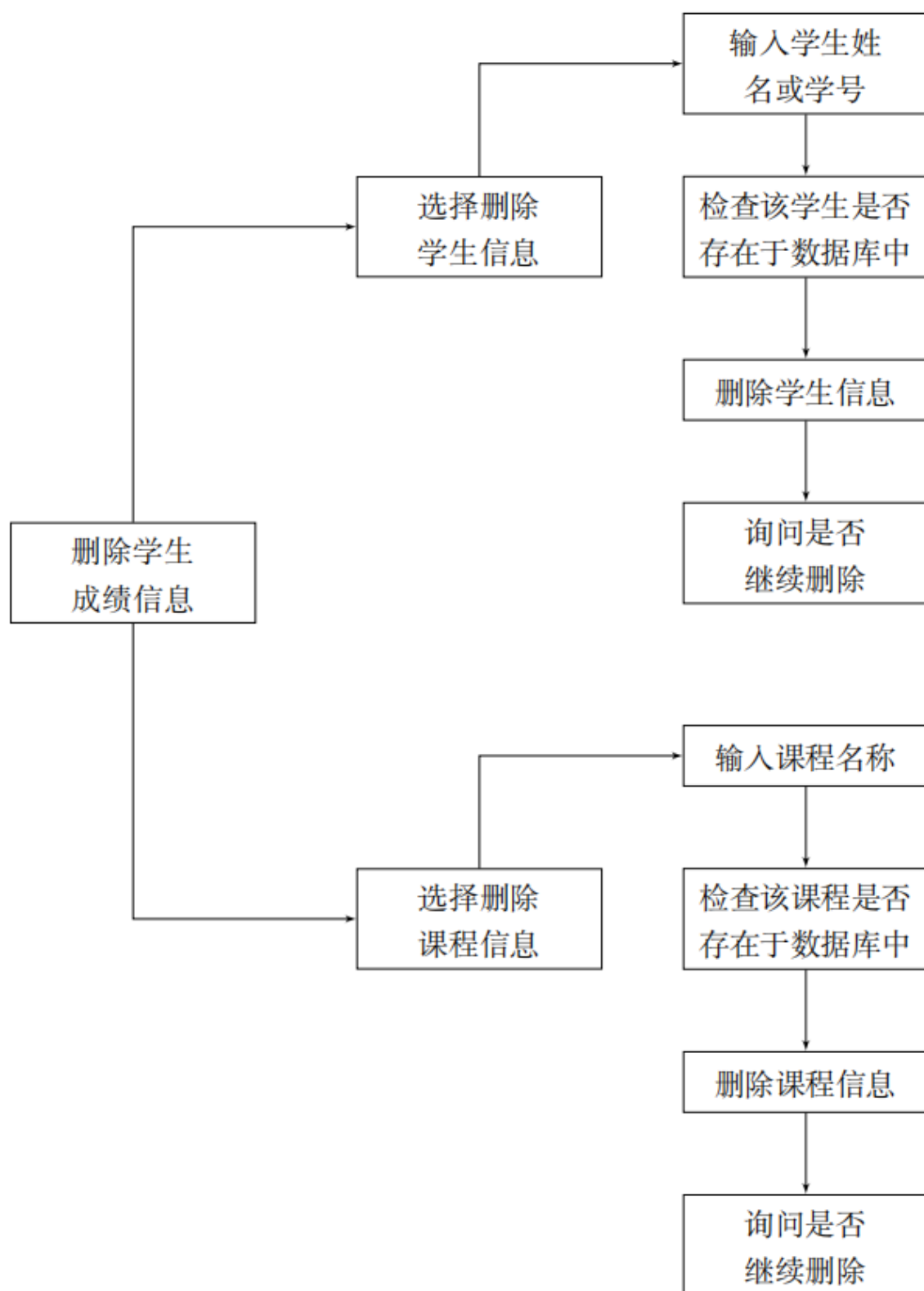


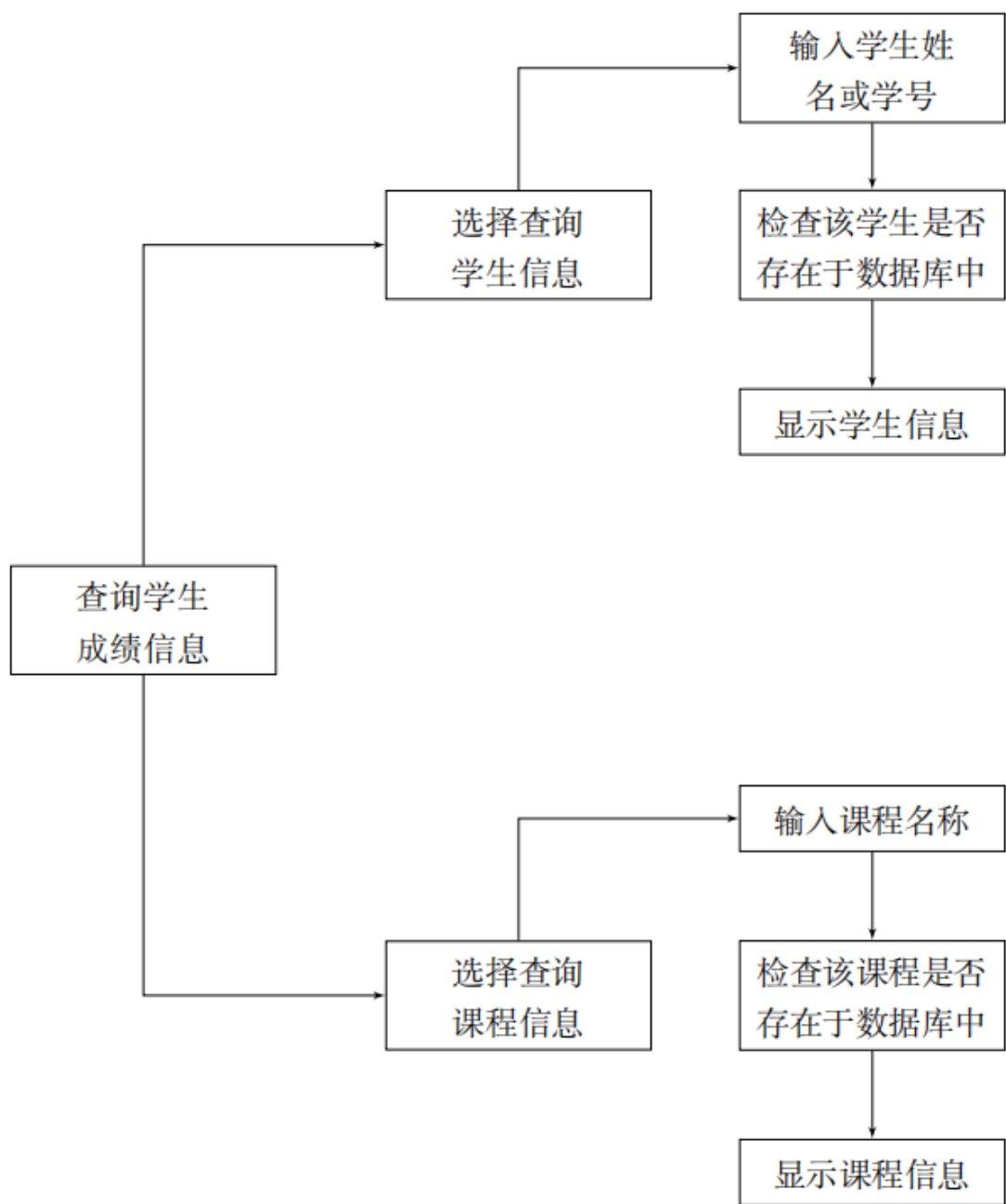
## 2. 学生成绩管理系统中各功能模块的实现

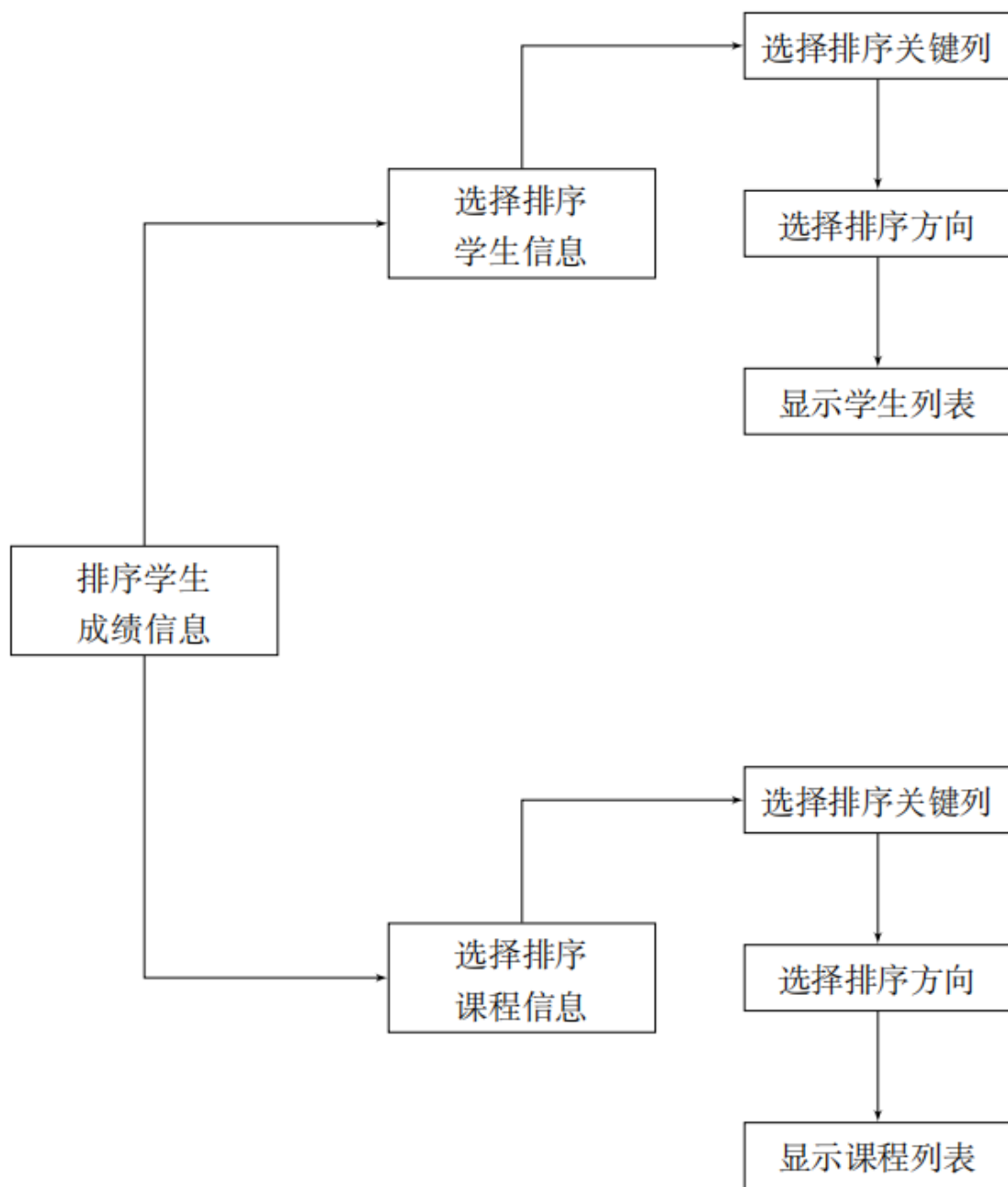


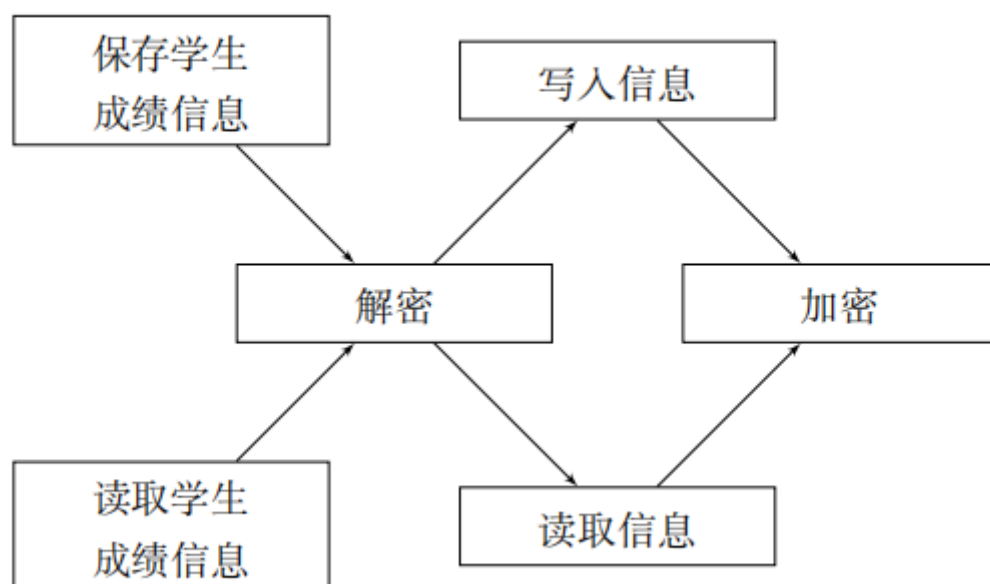
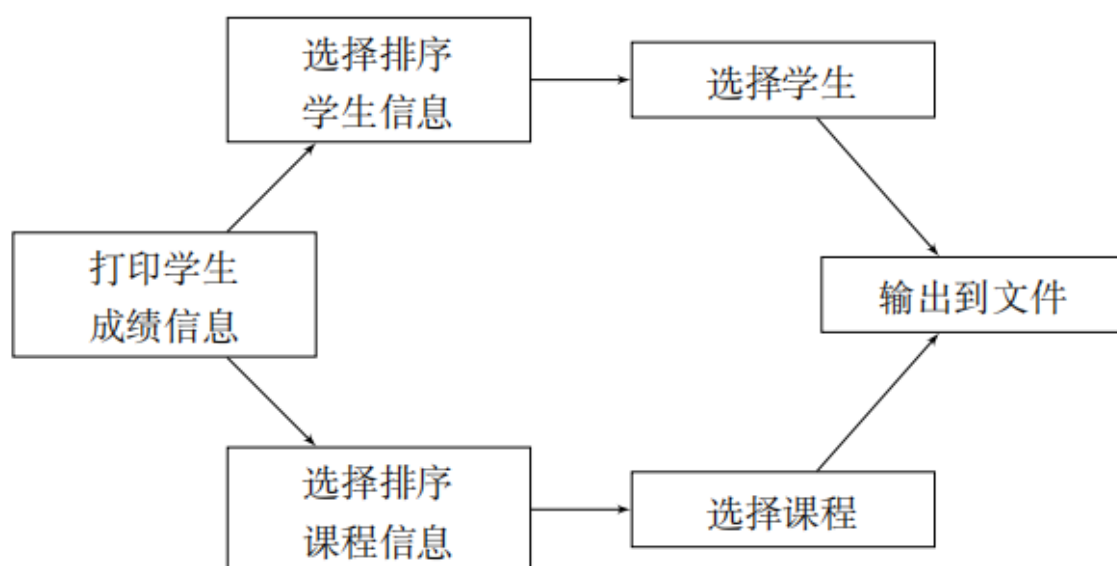












### 3. 类的UML图

若不清晰可参见[PDF\(纵向\)](#)或[PDF\(横向\)](#)。



## 五、系统调试

---

在“学生成绩管理系统”编写完成后，我对程序进行了系统调试。在调试过程中，我遇到了一些问题。

首先是键盘输入问题。起初，我使用 `cin` 来处理输入选项。但是当输入格式与设想的格式不符时，程序可能会出现众多异常情况。于是，我选择使用 `_getch()` 来直接读取键盘输入值，在一定程度上避免了麻烦。

文件保存上也遇到了一些问题。我为保存设计了加密算法，原理是对每个字符直接取反。但是在实际使用中，部分字符取反为文件终止符，导致读取失败。后来，我对算法进行了改进，现在已经可以正常在保存读取时加密、解密了。另外，我设想过通过密钥的方式进行加密，不过因为效果不是很理想，最终放弃了。

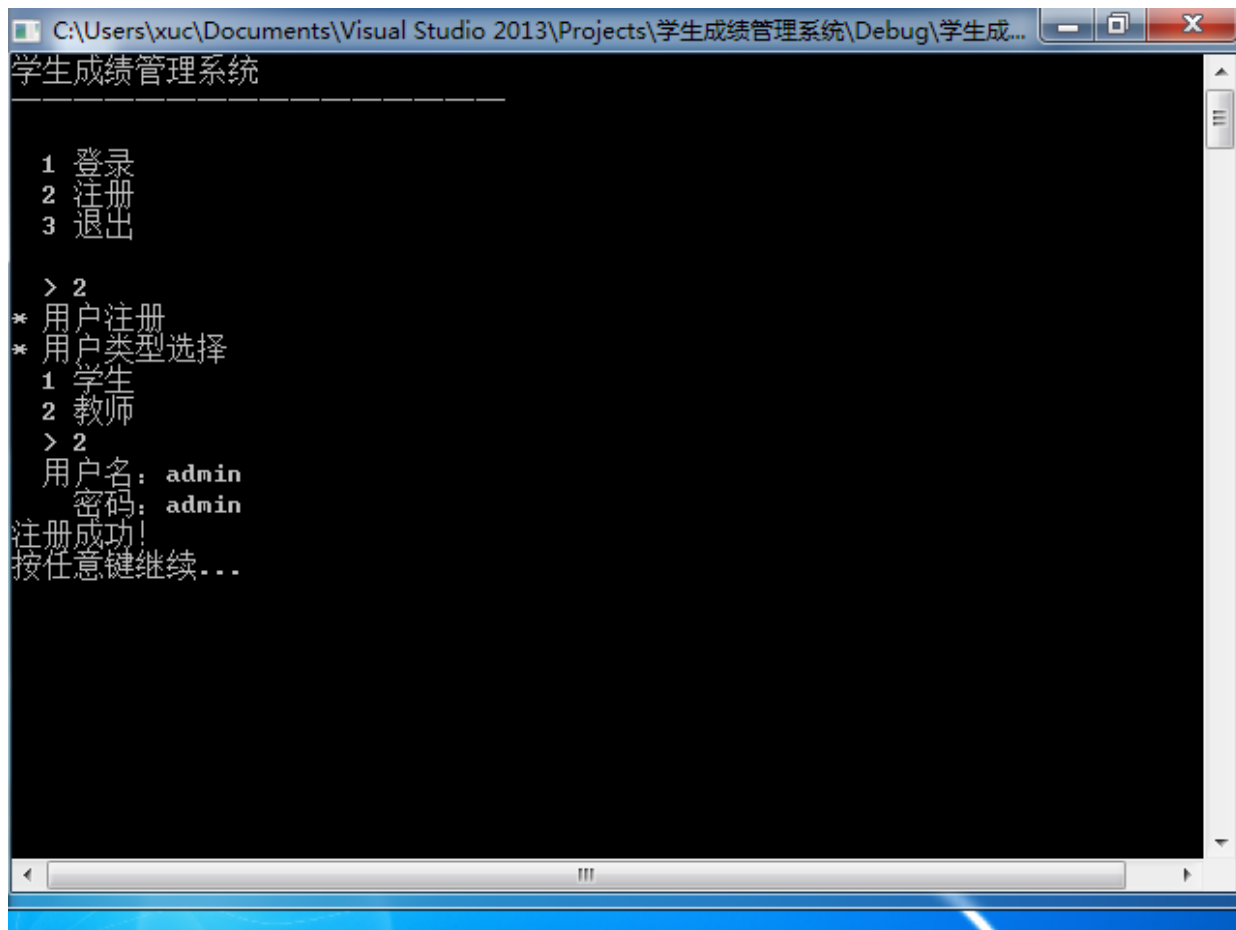
在数据存储形式上，我使用了比较传统的链表来存储数据。同时，我将学生成绩分为两份，同时存储在学生类和课程类中。这样做偶尔会出现删除不干净的情况，同时由于某一语句中括号放在了错误的位置，导致删除失败。后来我重新设计了将两边数据进行同步的算法，解决了这个问题。

总体而言，我的“学生成绩管理系统”的调试较为成功。虽然出现了一些小问题，但是基本都是因为我开发过程中的粗心和疏忽导致的，并不存在功能上面的缺陷。同时，由于我在一定程度上运用了模块化的编程思想，总体上解决问题也较为容易。

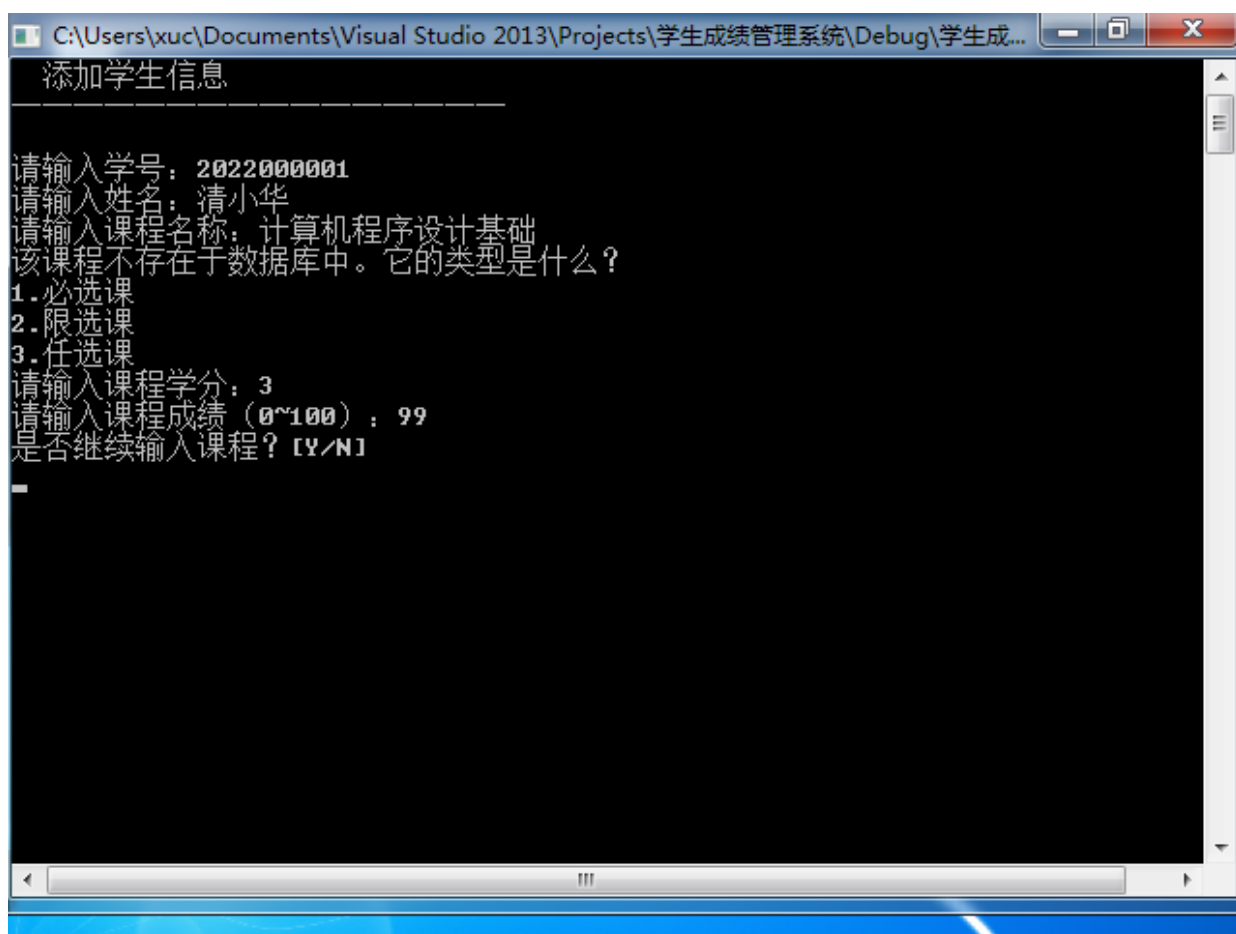
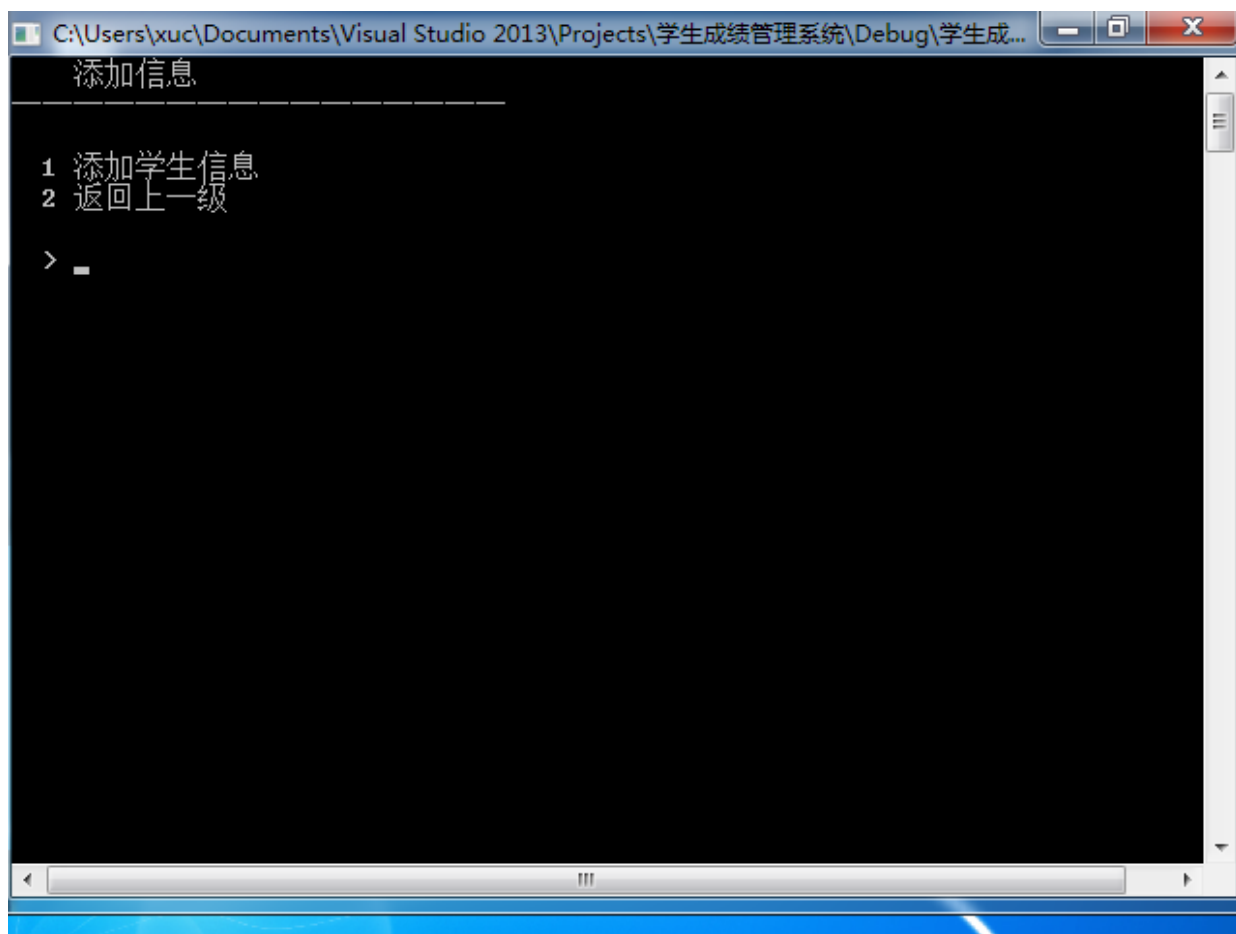
## 六、测试结果与分析

由于本程序在 UTF-8 (Codepage 65001) 编码环境下在输出一些含有汉字的表格时可能出现不对齐的情况，而我的电脑使用 UTF-8 编码环境，故以下截图是本程序运行在一台 Windows 7 虚拟机（使用 GBK (Codepage 936) 编码环境）上时的截图。

首先注册教师账户。



添加一名学生。





```
C:\Users\xuc\Documents\Visual Studio 2013\Projects\学生成绩管理系统\Debug\学生成...
该课程不存在于数据库中。它的类型是什么？
1.必修课
2.限选课
3.任选课
请输入课程学分：3
请输入课程成绩（0~100）：99
是否继续输入课程？[Y/N]
请输入课程名称：体育
该课程不存在于数据库中。它的类型是什么？
1.必修课
2.限选课
3.任选课
请输入课程学分：1
请输入课程成绩（0~100）：88
是否继续输入课程？[Y/N]
请输入课程名称：实验室科研探究
该课程不存在于数据库中。它的类型是什么？
1.必修课
2.限选课
3.任选课
请输入课程学分：1
请输入课程成绩（0~100）：77
该课程为任选课，得分是否计PF？[Y/N]
是否继续输入课程？[Y/N]
是否继续添加？[y/N]n
按任意键继续...
```

添加完成后查询到该学生，说明添加成功。

```
C:\Users\xuc\Documents\Visual Studio 2013\Projects\学生成绩管理系统\Debug\学生成...
查询学生信息
-----
1 按学号查询
2 按姓名查询
3 返回上一级

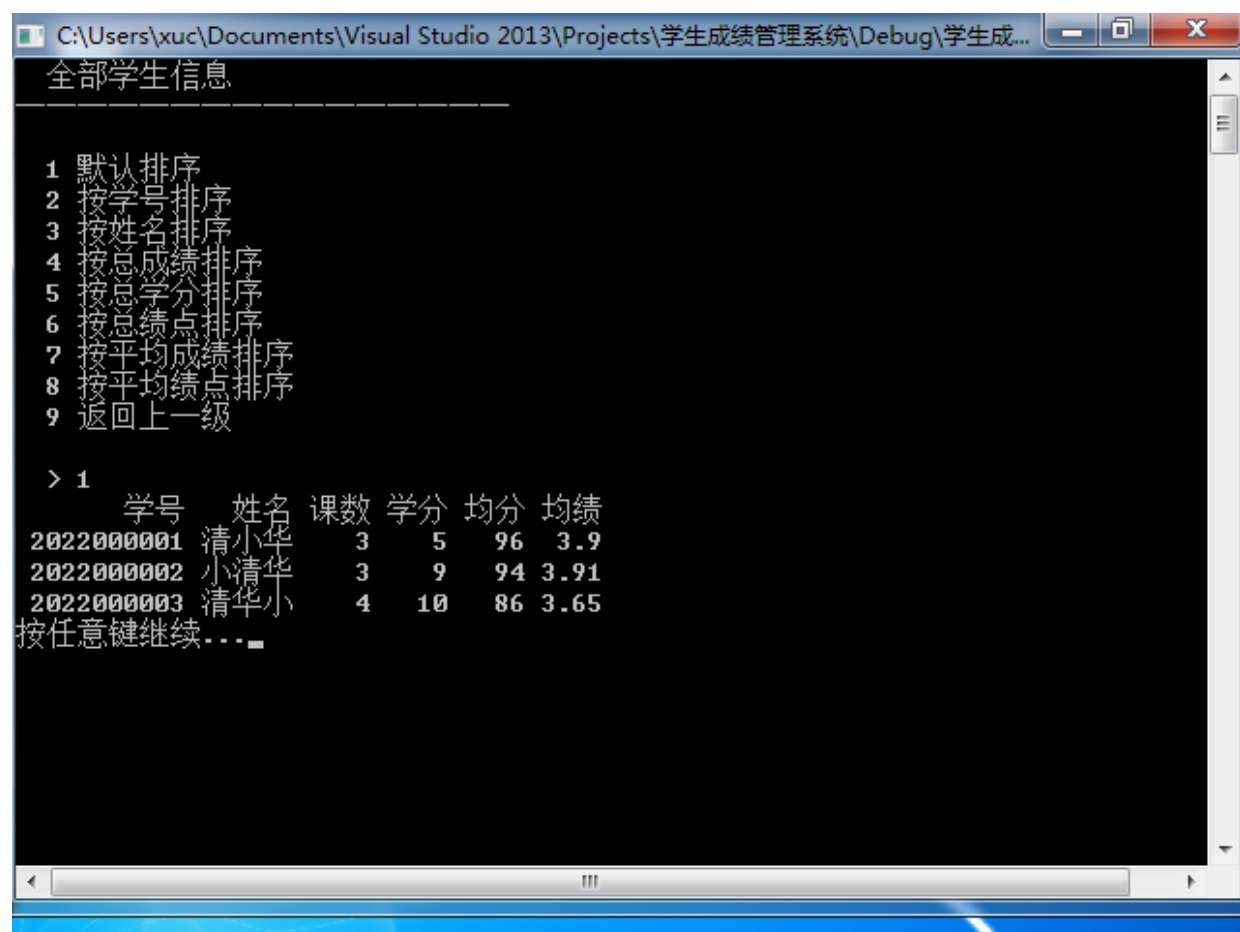
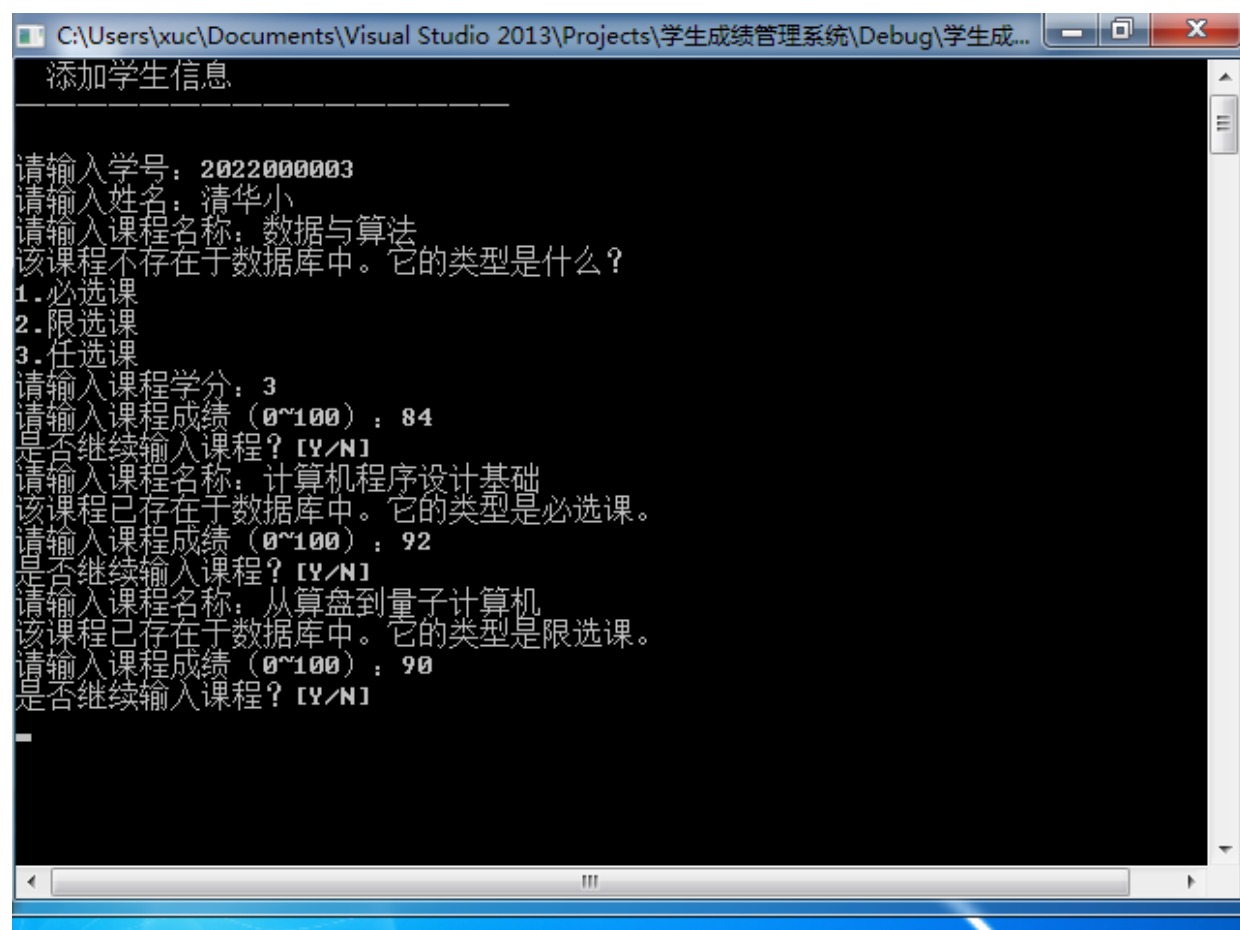
> 1
学号：2022000001

学号：2022000001
姓名：清小华
课程数：3
总学分：5
平均成绩：96
平均绩点：3.9

课程名称 类型 学分 成绩 绩点
计算机程序设计基础 必选 3 99 4
体育 限选 1 88 3.6
实验室科研探究 任选 1 N/A P

共找到1个学生。
按任意键继续...
```

继续添加一些学生。



```

C:\Users\xuc\Documents\Visual Studio 2013\Projects\学生成绩管理系统\Debug\学生成绩...
5 按平均分排序
6 按平均绩点排序
7 返回上一级

> 2

* 课程类型选择
1 全部课
2 必修课
3 限选课
4 任选课
> 1
* 排序方向选择
1 升序
2 降序
> 1

名称 类型 学分 人数 均分 均绩
从算盘到量子计算机 限选 2 2 88 3.6
电子电路与系统基础 必选 2 1 80 3.3
高等微积分 必选 5 1 98 4
计算机程序设计基础 必选 3 2 95 4
实验室科研探究 任选 1 1 0 0
数据与算法 必选 3 1 84 3.3
体育 限选 1 1 88 3.6
写作与沟通 任选 2 1 91 4
按任意键继续...

```

将学生成绩按平均绩点升序排序。

```

C:\Users\xuc\Documents\Visual Studio 2013\Projects\学生成绩管理系统\Debug\学生成绩...
全部学生信息
-----
1 默认排序
2 按学号排序
3 按姓名排序
4 按总成绩排序
5 按总学分排序
6 按总绩点排序
7 按平均成绩排序
8 按平均绩点排序
9 返回上一级

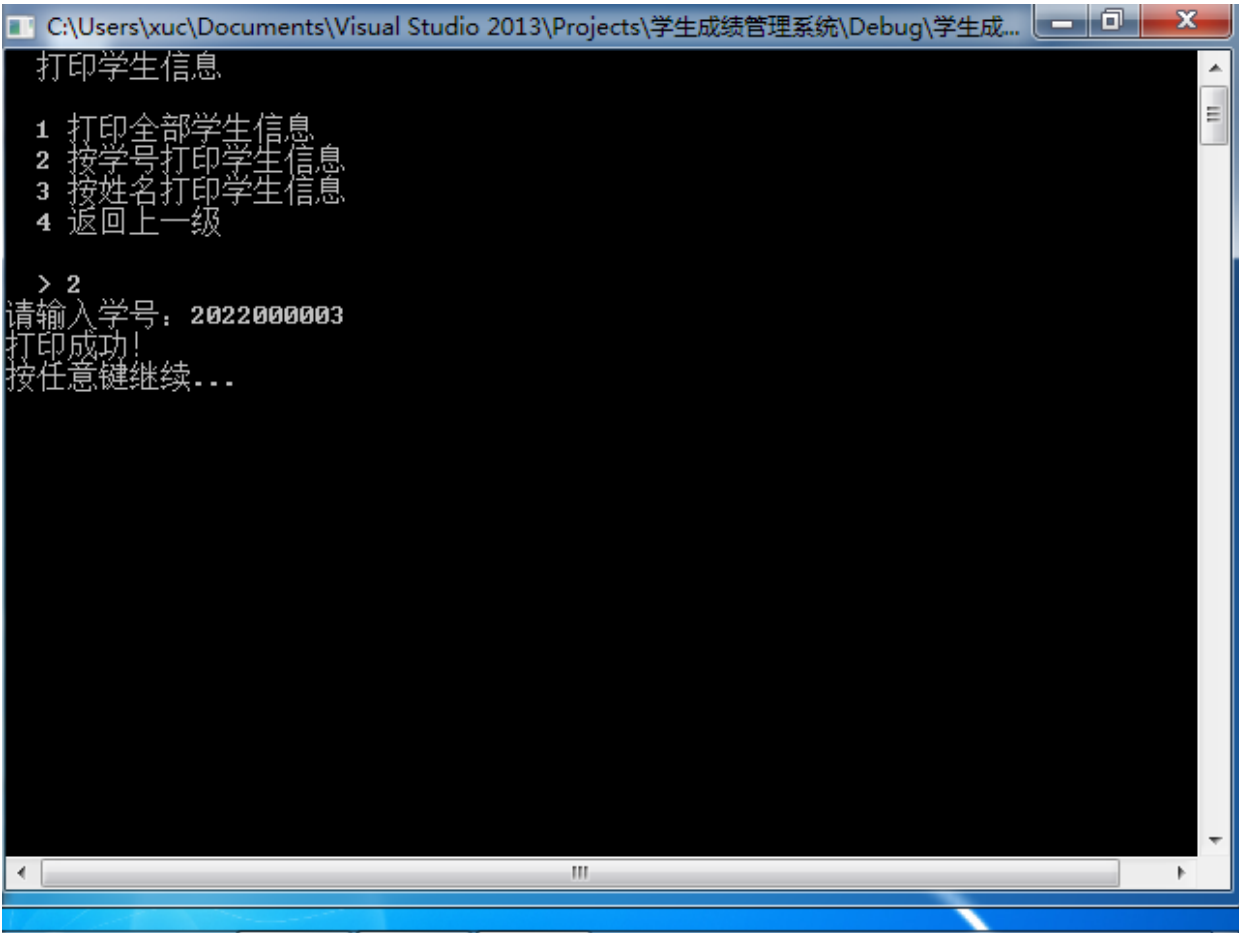
> 8

* 排序方向选择
1 升序
2 降序
> 1

学号 姓名 课数 学分 均分 均绩
2022000003 清华小 4 10 86 3.65
2022000001 清小华 3 5 96 3.9
2022000002 小清华 3 9 94 3.91
按任意键继续...

```

打印学生成绩。



# ××大学学生成绩单

学号：2022000003

姓名：清华小

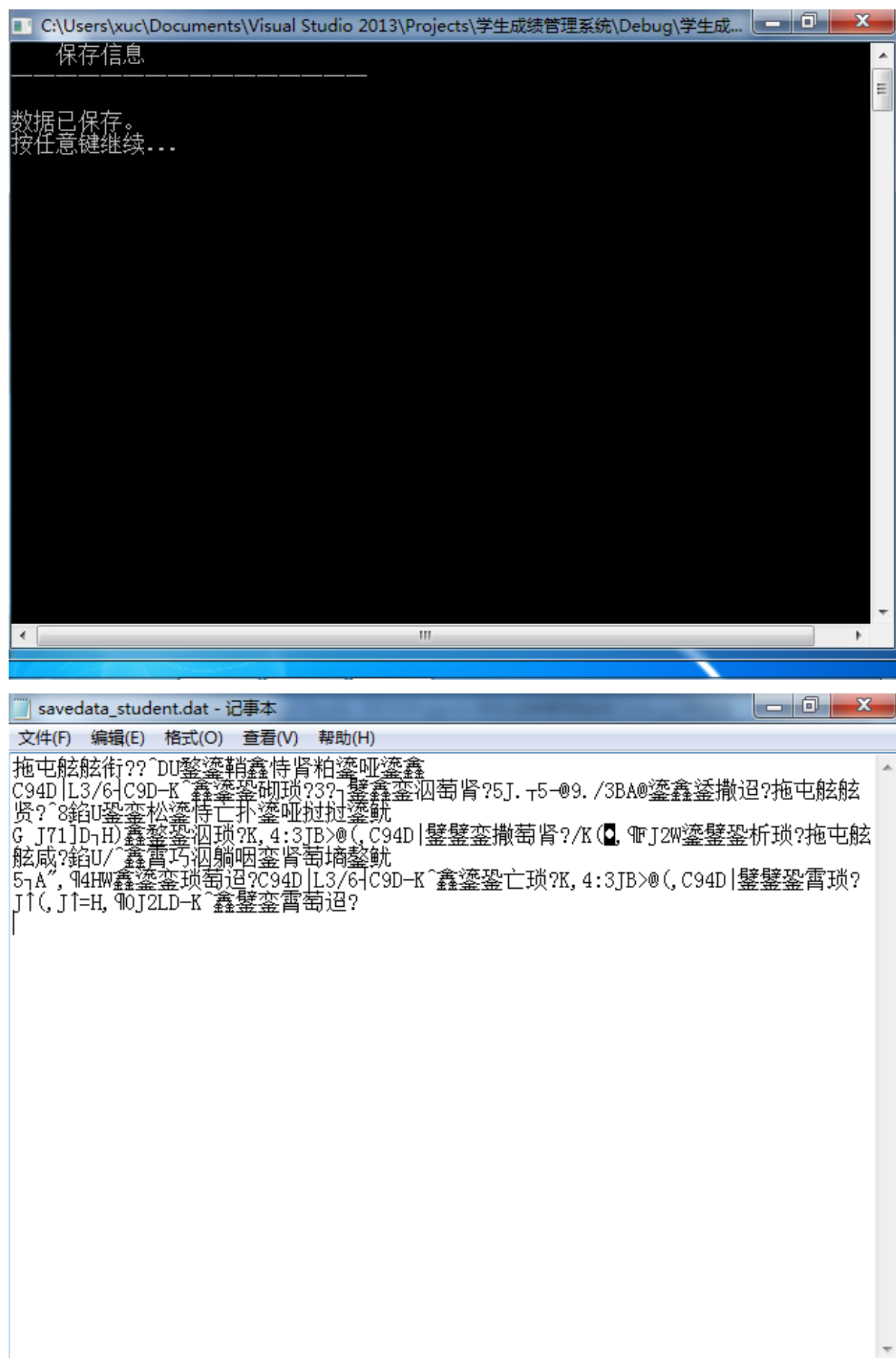
总学分：10

平均成绩：86

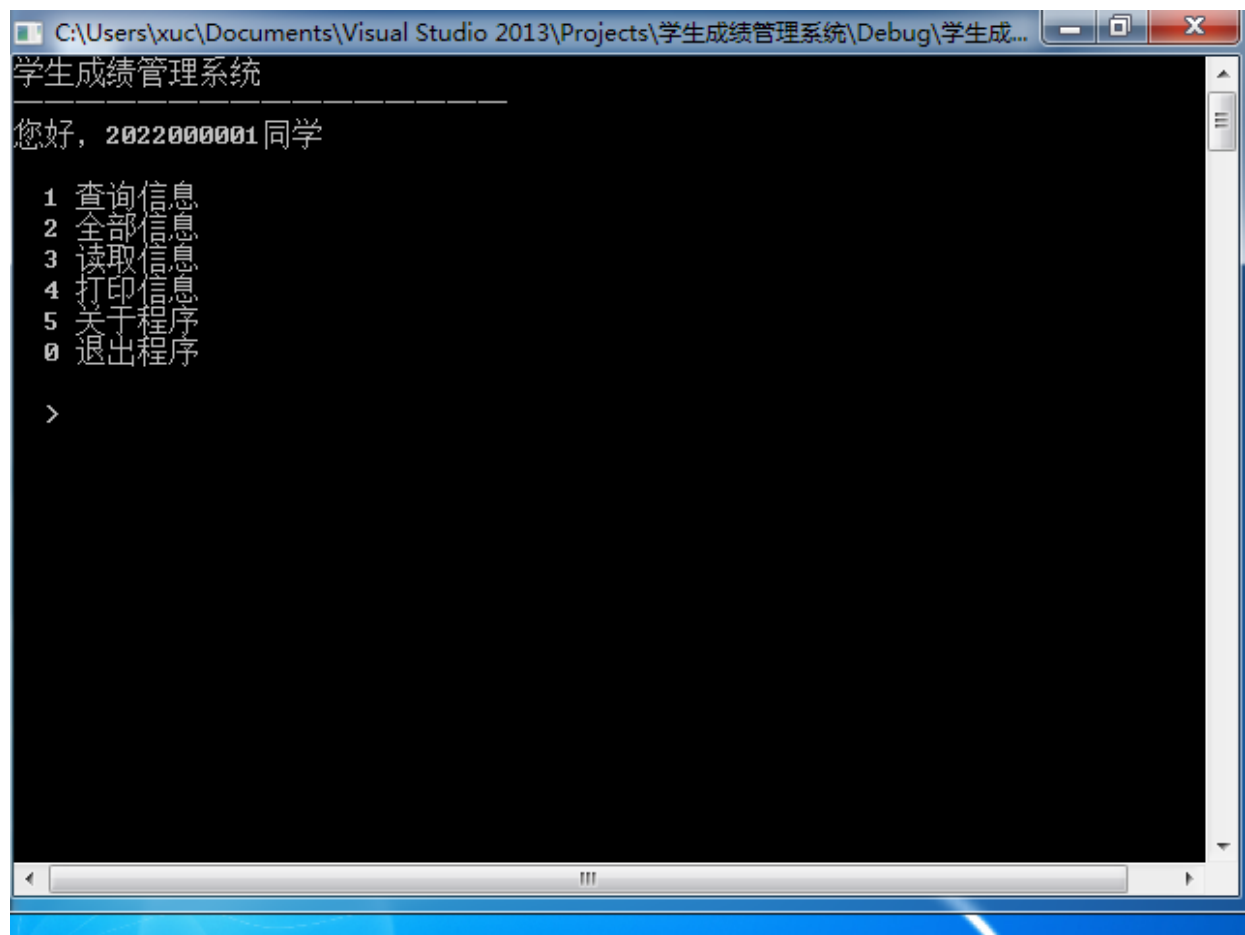
平均绩点：3.65

课程名称	课程类型	学分	成绩	绩点
数据与算法	必修课	3	84	3.3
计算机程序设计基础	必修课	3	92	4
从算盘到量子计算机	限选课	2	90	4
电子电路与系统基础	必修课	2	80	3.3

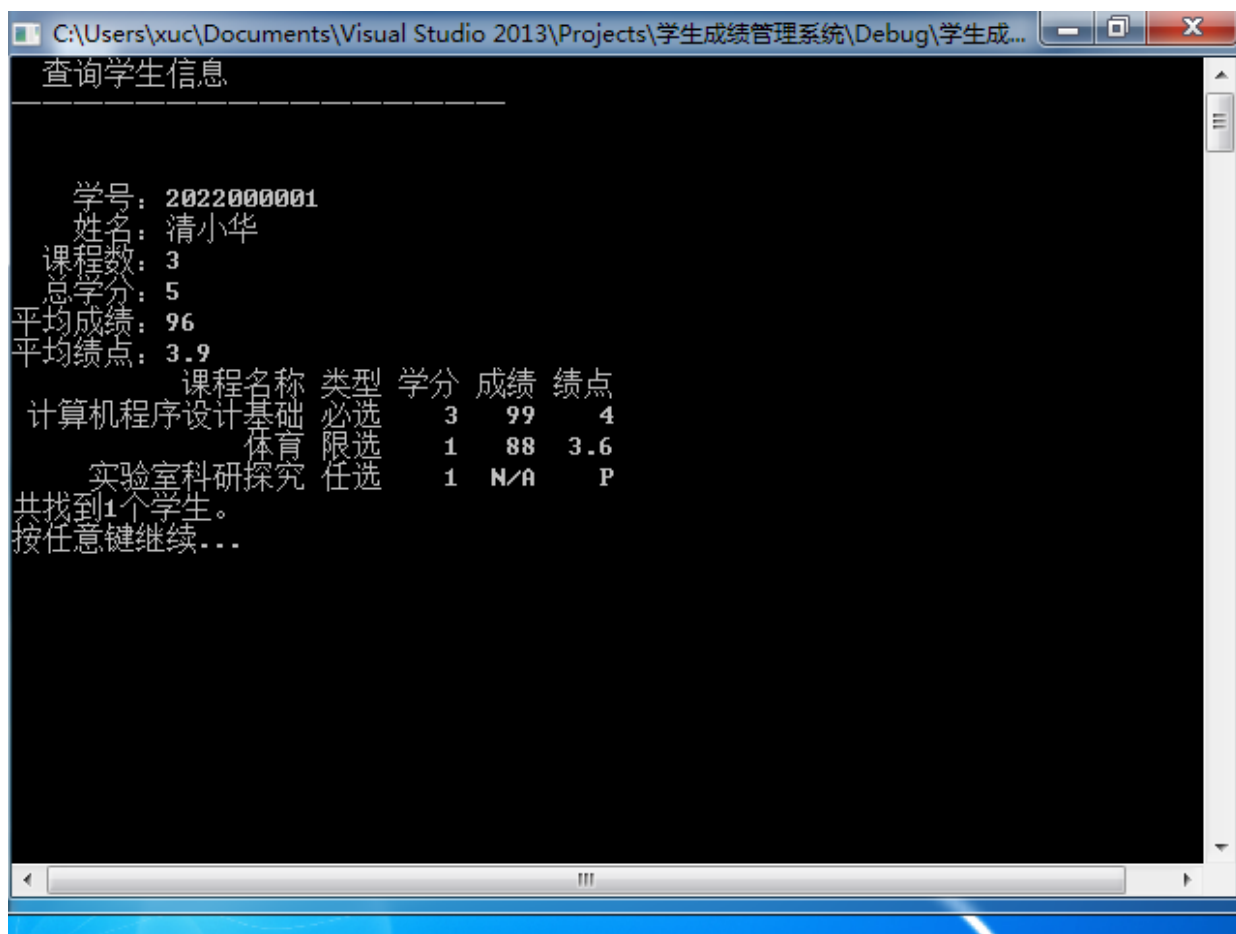
保存学生成绩，得到加密文件。



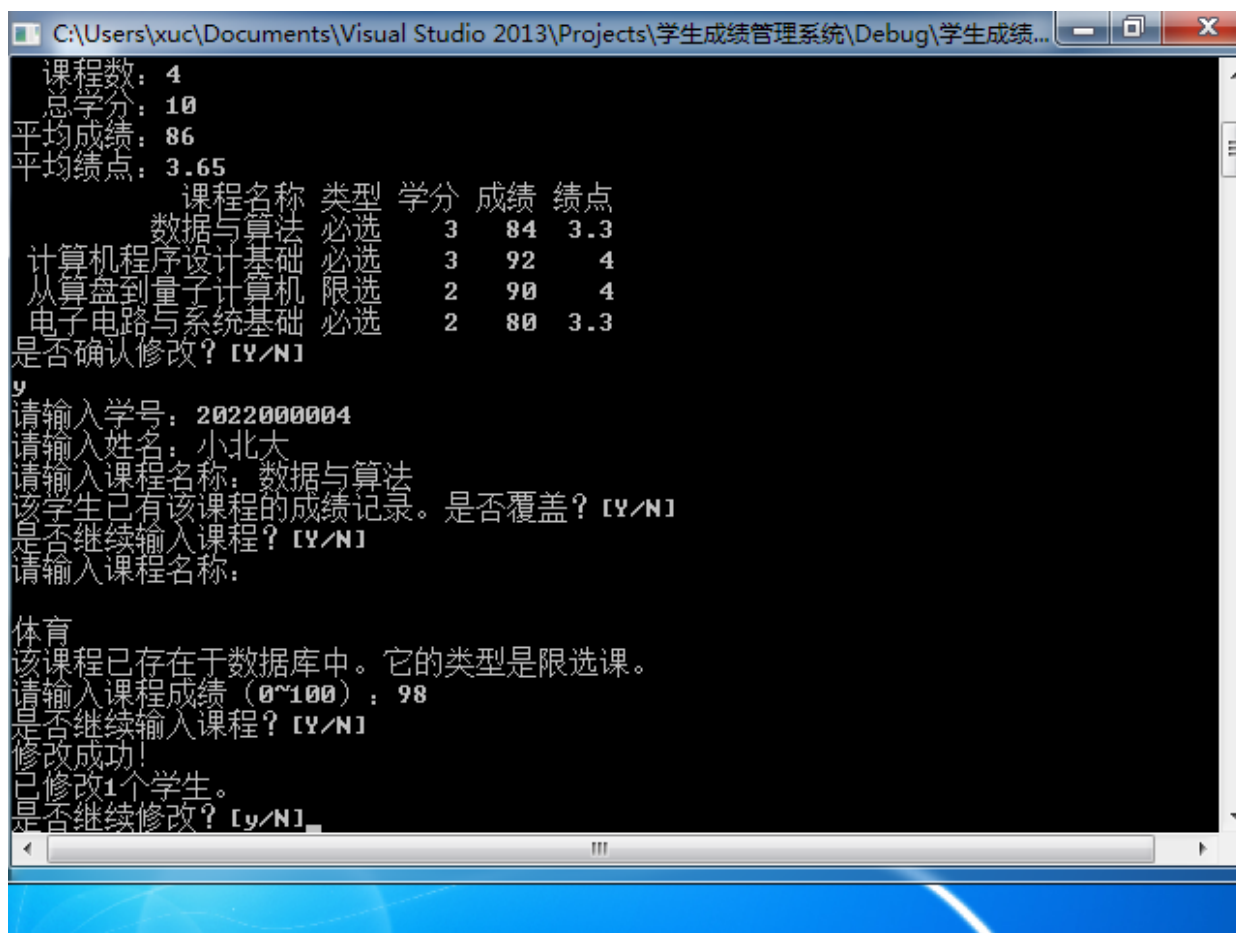
登录学生账户。



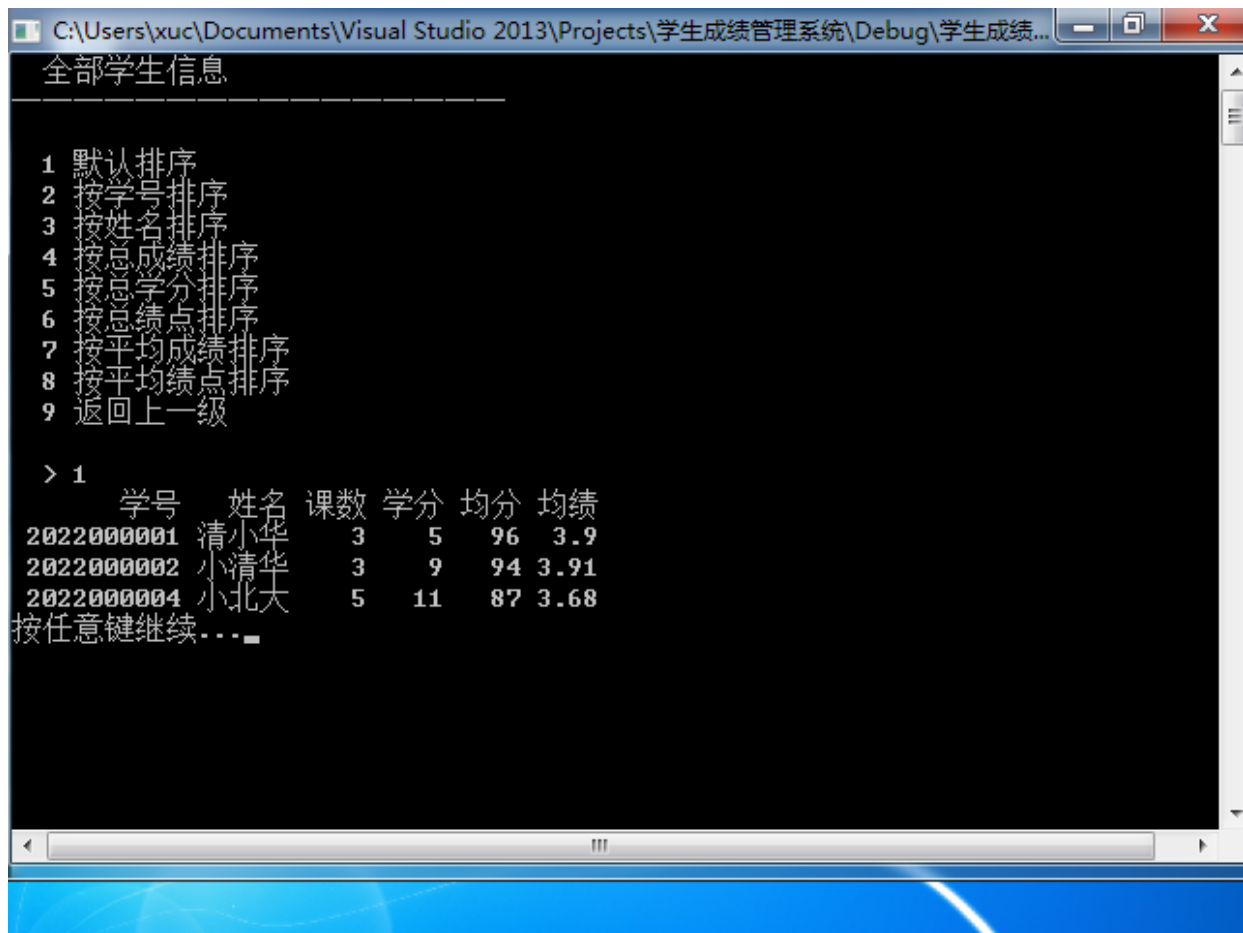
学生只可对自己的信息进行查看。



返回教师账号，修改学生信息。



成功修改。





## 七、总结

---

通过完成本次《计算机程序设计基础2》大作业，我对面向对象的程序设计、C++语言以及计算机与操作系统等多方面有了更加深入的认识。

在程序设计的过程中，我使用了大量的类和对象，在编写过程中比从春季学期的作业中更加深刻地体会到面向对象的编程思想。在设计类的时候，我尽可能使各个类的功能分开，将一些全局的方法设置在数据库类（`Database`）中，将不同的类的方法设置在相应的类中，在一定程度上实现了“高内聚低耦合”的设计原则。在程序运行的过程中，我充分使用了堕胎和虚函数的特性，赋予不同对象不同特性，提高了程序的灵活性和可扩展性。在测试过程中，我尝试使用了异常处理机制，增强程序安全性。

在具体方法上，我学习了一些C++语言的语法和特性，如模板、容器、迭代器、智能指针等，这些都是C++语言相比于C语言的优势所在。我尝试使用了一些STL标准库中的内容（如`string`、`vector`等），来对数据进行更好地管理，使得代码更加简洁高效、易于维护。

在计算机层面，我对文件操作有了更加深入的理解。我使用了文件流来读取和写入数据文件，实现了数据的持久化存储。同时，我采用了一种加密算法，并通过实践排除了其中的漏洞。除此之外，我还学习了使用`Git`、`LATEX`等工具帮助我进行程序编写与报告写作。这使我对计算机和程序设计理解更加深入。

总而言之，本次大作业使我对C++和计算机认识更加深入，程序设计水平得到一定程度提升。

当然，这次大作业也存在一些缺憾和不足。各个类之间仍然存在一定依赖性，没有完全达到封装的效果。另外，一些功能设计（如目前修改对象不能只针对一个特定的属性进行修改）仍然存在可改进的地方，如现在在程序运行时常常需要遍历所有学生和课程，造成资源浪费；同时，该程序只支持一个数据库，且没有为用户手动修改数据文件做好十分充分的准备。这可能主要是因为我最初拟定的计划不明确导致的。同时，如果本程序能够拥有图形化界面就更好了，然而我对`Qt`等不是很熟练，只好暂时搁置。

# 附录1：评分表

项 目	评 价	
设计方案的合理性与创新性	6	
设计与调试结果	8	
设计说明书的质量	2	
程序基本要求涵盖情况	8	
程序代码编写素养情况	4	
课程设计周表现情况	2	
综合成绩	30	

## 附录2：使用说明

---

若要输入含有汉字的名称，推荐使用 GBK (Codepage 936) 编码环境运行本程序。本程序在 UTF-8 (Codepage 65001) 编码环境下在输出一些含有汉字的表格时可能出现不对齐的情况，不过功能一切正常。

此程序虽然经过大量调试，但是仍然可能出现无法应对用户非正常输入的情况，运行本程序请遵照使用说明。

此程序开始运行时会自动读取存储在 `savedata_user.dat` 中的用户数据（包括用户名和密码）。该文件由程序自动生成并加密，应该尽量避免修改。

注册或登录账号后进入主菜单。若该用户为学生，则功能较少，且仅能查看与自己有关的信息，不能写入信息；若该用户为教师（管理员），则可添加信息、删除信息、修改信息，并可查看任意学生和课程信息。

进入主菜单后，学生与课程信息默认为空。故先需要读取存储在文件中的学生信息和课程信息。若读取成功，则数据库中有相关信息。需要注意的是，若读取时数据库中已存在学生或课程信息，则学生和课程信息会被清空。

数据库中存在相关信息后，即可对这些信息进行相应查询、打印等操作。学生的信息主要包含学号、姓名、选课信息、总学分、平均成绩、平均绩点等。课程的信息主要包含课程名称、课程学分、课程类型、选课学生信息、平均成绩、平均绩点等。其中，对于任选课，学生可以选择对该科成绩计“P/F”，从而不参加 GPA 的计算。

完成操作后，应及时保存信息。在退出程序时也会提示是否保存信息。

## 附录3：源程序清单

### 头文件

#### commonheader.h

```
1  #pragma once
2  #define _CRT_SECURE_NO_WARNINGS
3  #include <iostream>
4  #include <fstream>
5  #include <iomanip>
6  #include <string>
7  #include <vector>
8  #include <string.h>
9  #include <conio.h>
10 #include <time.h>
11
12 class Database;
13
14 enum LectureType
15 {
16     DEFAULT,
17     REQUIRED,
18     LIMITED,
19     OPTIONAL
20 };
21
22 class Info
23 {
24 protected:
25     // 名称 (name)、唯一标识符 (uid)、调试模式 (DebugMode)
26     std::string name;
27     int uid;
28     static int currentUid;
29     static bool DebugMode;
30
31 public:
32     Info();
33     Info(std::string inputName);
34     virtual ~Info() = 0;
35
36     virtual std::string getName();
37     int getUid();
38     static bool isDebugMode();
39
40     // virtual void updateInfo(Database& database) = 0;
41     virtual void setName(std::string inputName) = 0;
42     static void setDebug(bool inputDebugMode)
43     {
44         DebugMode = inputDebugMode;
```

```

45     }
46
47     virtual void printInfo(int) = 0;
48     virtual void printInfo(int, int) = 0;
49 };
50
51 class Account : virtual public Info
52 {
53 protected:
54     std::string password;
55
56 public:
57     Account();
58     Account(std::string inputName, std::string inputPassword);
59     ~Account();
60     virtual std::string getPassword();
61     virtual void setName(std::string inputName);
62     virtual void setPassword(std::string inputPassword);
63     virtual int getPermission() { return 0; }
64     virtual void printInfo(int) {}
65     virtual void printInfo(int, int) {}
66     friend std::istream& operator>>(std::istream&, Account&);
67     friend std::ostream& operator<<(std::ostream&, Account&);
68 };
69
70 class User : virtual public Account
71 {
72 public:
73     int getPermission() { return 1; }
74 };
75
76 class Admin : virtual public Account
77 {
78 public:
79     int getPermission() { return 2; }
80 };
81
82 // Student类, 对学生信息进行封装
83 class Student : virtual public Info, User
84 {
85 protected:
86     // 学号 (studentNo)、姓名 (name)、各科名称 (lectureName)、各科类型 (lectureType)、
87     // 各科成绩 (lectureScore)、各科学分 (lectureCredit)、学分绩点 (lectureGPA)、平均学分绩点
88     // (averageGPA)
89     int studentNo;
90     std::vector<std::string> lectureName;
91     std::vector<LectureType> lectureType;
92     std::vector<int> lectureScore;
93     std::vector<int> lectureCredit;
94     std::vector<double> lectureGPA;
95     std::vector<int> lecturePF;
96     int lectureNum;

```

```
95     int lecturePFNum;
96     int totalScore;
97     int totalCredit;
98     int totalPFCredit;
99     double totalGPA;
100    double averageScore;
101    double averageGPA;
102
103    public:
104        // 构造函数
105        Student();
106
107        // 析构函数
108        ~Student();
109
110        // 访问数据成员的接口函数
111        int getStudentNo();
112        std::string getName();
113        std::vector<std::string> getLectureName();
114        std::vector<LectureType> getLectureType();
115        std::vector<int> getLectureScore();
116        std::vector<int> getLectureCredit();
117        std::vector<double> getLectureGPA();
118        std::vector<int> getLecturePF();
119        int getLectureNum();
120        int getLecturePFNum();
121        int getTotalScore();
122        int getTotalCredit();
123        int getTotalPFCredit();
124        double getTotalGPA();
125        double getAverageScore();
126        double getAverageGPA();
127
128        // 修改数据成员的接口函数
129        void updateInfo(Database &database);
130        void setStudentNo(int inputStudentNo);
131        void setName(std::string inputName);
132        void setLectureName(std::vector<std::string> inputLectureName);
133        void setLectureType(std::vector<LectureType> inputLectureType);
134        void setLectureScore(std::vector<int> inputLectureScore);
135        void setLectureCredit(std::vector<int> inputLectureCredit);
136        void setLectureGPA(std::vector<double> inputLectureGPA);
137        void setLecturePF(std::vector<int> inputLecturePF);
138        void setLectureNum(int inputLectureNum);
139        void setLecturePFNum(int inputLecturePFNum);
140        void setTotalScore(int inputTotalScore);
141        void setTotalCredit(int inputTotalCredit);
142        void setTotalPFCredit(int inputTotalPFCredit);
143        void setTotalGPA(double inputTotalGPA);
144        void setAverageScore(double inputAverageScore);
145        void setAverageGPA(double inputAverageGPA);
146
```

```

147 // 打印学生信息
148 void printInfo(int) {}
149 void printInfo(int widthStudentNo, int widthName);
150 void printStudentInfo();
151
152 // 运算符重载
153 friend std::istream &operator>>(std::istream &, Student &);
154 friend std::ostream &operator<<(std::ostream &, Student &);
155 };
156
157 class Lecture : virtual public Info
158 {
159 protected:
160     // 课程名称 (name)、课程学分 (credit)
161     int credit;
162     std::vector<int> studentNo;
163     std::vector<std::string> studentName;
164     std::vector<int> studentScore;
165     std::vector<double> studentGPA;
166     int studentNum;
167     int totalScore;
168     double totalGPA;
169     int averageScore;
170     double averageGPA;
171
172 public:
173     // 构造函数
174     Lecture();
175
176     // 析构函数
177     virtual ~Lecture();
178
179     // 访问数据成员的接口函数
180     virtual LectureType getLectureType() { return DEFAULT; }
181     virtual int getCredit();
182     virtual std::vector<int> getStudentNo();
183     virtual std::vector<std::string> getStudentName();
184     virtual std::vector<int> getStudentScore();
185     virtual std::vector<double> getStudentGPA();
186     virtual int getStudentNum();
187     virtual int getTotalScore();
188     virtual double getTotalGPA();
189     virtual int getAverageScore();
190     virtual double getAverageGPA();
191
192     // 设置数据成员的接口函数
193     virtual void updateInfo(Database &database);
194     virtual void setName(std::string inputName);
195     virtual void setCredit(int inputCredit);
196     virtual void setStudentNo(std::vector<int> inputStudentNo);
197     virtual void setStudentName(std::vector<std::string> inputStudentName);
198     virtual void setStudentScore(std::vector<int> inputStudentScore);

```

```

199     virtual void setStudentGPA(std::vector<double> inputStudentGPA);
200     virtual void setStudentNum(int inputStudentNum);
201     virtual void setTotalScore(int inputTotalScore);
202     virtual void setTotalGPA(double inputTotalGPA);
203     virtual void setAverageScore(int inputAverageScore);
204     virtual void setAverageGPA(double inputAverageGPA);
205     virtual void addStudent(int inputStudentNo, std::string inputStudentName, int
inputStudentScore, double inputStudentGPA);

206
207     // 打印课程信息
208     virtual void printInfo(int widthName);
209     virtual void printInfo(int, int) {}
210     virtual void printLectureInfo();
211
212     // 运算符重载
213     friend std::istream &operator>>(std::istream &, Lecture &);
214     friend std::ostream &operator<<(std::ostream &, Lecture &);
215 };
216
217 class LectureRequired : virtual public Lecture
218 {
219 protected:
220 public:
221     // 构造函数
222     LectureRequired();
223
224     // 析构函数
225     ~LectureRequired();
226
227     // 访问数据成员的接口函数
228     virtual LectureType getLectureType() { return REQUIRED; }
229
230     // 设置数据成员的接口函数
231     void updateInfo(Database &database);
232
233     // 打印课程信息
234     void printInfo(int widthName);
235     void printInfo(int, int) {}
236     void printLectureInfo();
237
238     // 运算符重载
239     friend std::istream &operator>>(std::istream &, LectureRequired &);
240     friend std::ostream &operator<<(std::ostream &, LectureRequired &);
241 };
242
243 class LectureLimited : virtual public Lecture
244 {
245 protected:
246 public:
247     // 构造函数
248     LectureLimited();
249

```



```

250 // 析构函数
251 ~LectureLimited();
252
253 // 访问数据成员的接口函数
254 virtual LectureType getLectureType() { return LIMITED; }
255
256 // 设置数据成员的接口函数
257 void updateInfo(Database &database);
258
259 // 打印课程信息
260 void printInfo(int widthName);
261 void printInfo(int, int) {}
262 void printLectureInfo();
263
264 // 运算符重载
265 friend std::istream &operator>>(std::istream &, LectureLimited &);
266 friend std::ostream &operator<<(std::ostream &, LectureLimited &);
267 };
268
269 class LectureOptional : virtual public Lecture
270 {
271 protected:
272     std::vector<int> studentPF;
273     int studentPFNum;
274
275 public:
276     // 构造函数
277     LectureOptional();
278
279     // 析构函数
280     ~LectureOptional();
281
282     // 访问数据成员的接口函数
283     virtual LectureType getLectureType() { return OPTIONAL; }
284     std::vector<int> getStudentPF();
285     int getStudentPFNum();
286
287     // 设置数据成员的接口函数
288     void setStudentPF(std::vector<int> inputStudentPF);
289     void setStudentPFNum(int inputStudentPFNum);
290     void updateInfo(Database &database);
291     virtual void addStudent(int inputStudentNo, std::string inputStudentName, int
inputStudentScore, double inputStudentGPA, bool inputStudentPF);
292
293     // 打印课程信息
294     void printInfo(int widthName);
295     void printInfo(int, int) {}
296     void printLectureInfo();
297
298     // 运算符重载
299     friend std::istream &operator>>(std::istream &, LectureOptional &);
300     friend std::ostream &operator<<(std::ostream &, LectureOptional &);

```

```

301 };
302
303 // 基于模板的链表结点定义
304 template <class T>
305 struct Node
306 {
307     T data;
308     Node<T> *next;
309 };
310
311 // 基于模板的链表类声明
312 template <class T>
313 class LinkedList
314 {
315 private:
316     Node<T> *head; // 头结点
317     Node<T> *current; // 当前结点
318     inline void deepCopy(const LinkedList<T> &original); // 内联函数，用于深拷贝
319 public:
320     // 构造函数、复制构造函数、析构函数
321     LinkedList();
322     LinkedList(const LinkedList<T> &aplist);
323     ~LinkedList();
324
325     void insert(Node<T> *newNode); // 在头部之前插入元素
326     void insert_end(Node<T> *newNode); // 在尾部插入
327     Node<T> *getFirst(); // 获得链表头的数据
328     inline Node<T> *getNext(); // 获得当前结点的下一个数据
329     bool find(const T &element); // 查找一个数据
330     bool retrieve(T &element); // 检索一个数据
331     bool replace(const T &newElement); // 替换一个数据
332     bool remove(Node<T> *node); // 移除一个数据
333     bool isEmpty() const; // 判断链表是否为空
334     void makeEmpty(); // 清空链表
335     int size(); // 获得链表的大小
336
337     // 重载"="运算符
338     LinkedList<T> &operator=(const LinkedList<T> &rlist);
339 };
340
341 // 无参构造函数
342 template <class T>
343 LinkedList<T>::LinkedList()
344 {
345     head = current = nullptr;
346 }
347
348 // 复制构造函数
349 template <class T>
350 LinkedList<T>::LinkedList(const LinkedList<T> &aplist)
351 {
352     deepCopy(aplist);

```

```
353     }
354
355     // 析构函数
356     template <class T>
357     LinkedList<T>::~LinkedList()
358     {
359         makeEmpty();
360     }
361
362     // 在头部之前插入函数
363     template <class T>
364     void LinkedList<T>::insert(Node<T> *newNode)
365     {
366         current = nullptr;
367         newNode->next = head;
368         head = newNode;
369     }
370
371     // 在尾部之后插入函数
372     template <class T>
373     void LinkedList<T>::insert_end(Node<T> *newNode)
374     {
375         current = nullptr;
376         Node<T> *tail = head;
377         newNode->next = nullptr;
378         if (tail == nullptr)
379         {
380             head = newNode;
381         }
382         else
383         {
384             while (tail->next != nullptr)
385             {
386                 tail = tail->next;
387             }
388             tail->next = newNode;
389         }
390     }
391
392     // 获得链表头的函数
393     template <class T>
394     Node<T> *LinkedList<T>::getFirst()
395     {
396         if (head == nullptr)
397         {
398             return nullptr;
399         }
400         current = head;
401         return head;
402     }
403
404     // 获得下一个数据
```

```

405 template <class T>
406 Node<T> *LinkedList<T>::getNext()
407 {
408     if (current == nullptr)
409     {
410         return nullptr;
411     }
412     if (current->next == nullptr)
413     {
414         current = nullptr;
415         return nullptr;
416     }
417     current = current->next;
418     return current;
419 }
420
421 // 查找一个数据
422 template <class T>
423 bool LinkedList<T>::find(const T &element)
424 {
425     Node<T> *n;
426     n = getFirst();
427     if (n == nullptr)
428     {
429         return false;
430     }
431     do
432     {
433         if (n->data == element)
434         {
435             return true;
436         }
437         n = getNext();
438     } while (n != nullptr);
439     return false;
440 }
441
442 // 检索一个数据
443 template <class T>
444 bool LinkedList<T>::retrieve(T &element)
445 {
446     if (!find(element))
447     {
448         return false;
449     }
450     element = current->data;
451     return true;
452 }
453
454 // 替换一个数据
455 template <class T>
456 bool LinkedList<T>::replace(const T &newElement)

```

```

457 {
458     if (current == nullptr)
459     {
460         return false;
461     }
462     current->data = newElement;
463     return true;
464 }
465
466 // 移除一个数据
467 template <class T>
468 bool LinkedList<T>::remove(Node<T> *n)
469 {
470     current = nullptr;
471     if (head == nullptr)
472     {
473         return false;
474     }
475     Node<T> *tmp = head;
476     if (tmp == n)
477     {
478         head = tmp->next;
479         delete tmp;
480         return true;
481     }
482     while (tmp->next != nullptr)
483     {
484         if (tmp->next == n)
485         {
486             Node<T> *ptr = tmp->next;
487             tmp->next = ptr->next;
488             delete ptr;
489             return true;
490         }
491         tmp = tmp->next;
492     }
493     return false;
494 }
495
496 // 判断是否为空
497 template <class T>
498 bool LinkedList<T>::isEmpty() const
499 {
500     return head == nullptr;
501 }
502
503 // 将链表清空
504 template <class T>
505 void LinkedList<T>::makeEmpty()
506 {
507     while (head != nullptr)
508     {

```

```

509         current = head;
510         head = head->next;
511         delete current;
512     }
513     current = nullptr;
514 }
515
516 // 获得链表大小
517 template <class T>
518 int LinkedList<T>::size()
519 {
520     int size = 0;
521     Node<T> *tmp = head;
522     while (tmp != nullptr)
523     {
524         size++;
525         tmp = tmp->next;
526     }
527     return size;
528 }
529
530 // "="运算符重载
531 template <class T>
532 LinkedList<T> &LinkedList<T>::operator=(const LinkedList<T> &rlist)
533 {
534     if (this == &rlist)
535     {
536         return *this;
537     }
538     makeEmpty();
539     deepCopy(rlist);
540     return *this;
541 }
542
543 // 深拷贝函数
544 template <class T>
545 void LinkedList<T>::deepCopy(const LinkedList<T> &original)
546 {
547     head = current = nullptr;
548     if (original.head == nullptr)
549     {
550         return;
551     }
552     Node<T> *copy = head = new Node<T>;
553     Node<T> *origin = original.head;
554     copy->data = origin->data;
555     if (origin == original.current)
556     {
557         current = copy;
558     }
559     while (origin->next != nullptr)
560     {

```

```

561     copy->next = new Node<T>;
562     origin = origin->next;
563     copy = copy->next;
564     copy->data = origin->data;
565     if (origin == original.current)
566     {
567         current = copy;
568     }
569 }
570 copy->next = nullptr;
571 }
572
573 class FileException
574 {
575 public:
576     std::string filename;
577     std::string mode;
578     std::string type;
579
580     FileException(std::string inputFilename, std::string inputMode, std::string
inputType)
581     {
582         filename = inputFilename;
583         mode = inputMode;
584         type = inputType;
585     };
586
587     ~FileException(){};
588 };
589
590 // 数据库类，对程序使用的数据结构进行封装
591 class Database
592 {
593 private:
594     LinkedList<Student> studentList;
595     LinkedList<LectureRequired> requiredList;
596     LinkedList<LectureLimited> limitedList;
597     LinkedList<LectureOptional> optionalList;
598     LinkedList<User> userList;
599     LinkedList<Admin> adminList;
600
601 public:
602     // 构造函数与析构函数
603     Database();
604     ~Database();
605
606     // 列表相关函数
607     int getStudentListSize();
608     int getRequiredListSize();
609     int getLimitedListSize();
610     int getOptionalListSize();
611

```

```

612 // 文件读写函数
613 void load();
614 void save();
615
616 void loadStudent(const std::string &filename);
617 void loadRequired(const std::string &filename);
618 void loadLimited(const std::string &filename);
619 void loadOptional(const std::string &filename);
620
621 void saveStudent(const std::string &filename);
622 void saveRequired(const std::string &filename);
623 void saveLimited(const std::string &filename);
624 void saveOptional(const std::string &filename);
625
626 void encrypt(const std::string &filename);
627 void encrypt_key(const std::string &filename, const std::string &keyFilename);
628 void key_gen(const std::string &filename);
629 std::string timeStampToString(const time_t &timeStamp);
630
631 // 删除学生与课程函数
632 bool deleteStudent(const std::string &name);
633 bool deleteStudent(int studentNo);
634 bool deleteRequired(const std::string &name);
635 bool deleteLimited(const std::string &name);
636 bool deleteOptional(const std::string &name);
637
638 // 添加学生与课程函数
639 void addStudent();
640 void addRequired(const std::string &name, int credit);
641 void addLimited(const std::string &name, int credit);
642 void addOptional(const std::string &name, int credit);
643 void addStudentToLecture(const std::string &name, LectureType type, Student &stu);
644
645 //// 显示学生与课程函数
646 // void displayStudent(const std::string& name);
647 // void displayStudent(int studentNo);
648 // void displayLecture(const std::string& name);
649 // void displayRequired(const std::string& name);
650 // void displayLimited(const std::string& name);
651 // void displayOptional(const std::string& name);
652 // int displayAllStudent();
653 // int displayAllLecture(LectureType type);
654
655 // 修改学生与课程函数
656 bool modifyStudent(const std::string &name);
657 bool modifyStudent(int studentNo);
658 bool modifyLecture(const std::string &name);
659 bool modifyRequired(const std::string &name);
660 bool modifyLimited(const std::string &name);
661 bool modifyOptional(const std::string &name);
662
663 // 查询学生与课程函数

```



```

664     int queryStudent(const std::string &name, bool display = true);
665     int queryStudent(int studentNo, bool display = true);
666     int queryLecture(const std::string &name, bool display = true);
667     int queryRequired(const std::string &name, bool display = true);
668     int queryLimited(const std::string &name, bool display = true);
669     int queryOptional(const std::string &name, bool display = true);
670     Student *findStudent(const std::string &name);
671     Student *findStudent(int studentNo);
672     Lecture *findLecture(const std::string &name);
673     LectureRequired *findRequired(const std::string &name);
674     LectureLimited *findLimited(const std::string &name);
675     LectureOptional *findOptional(const std::string &name);
676
677     // 排序函数
678     void sortStudent(int direction, int keycol);
679     void sortLecture(LectureType type, int direction, int keycol);
680     void sortStudentCustom(Student **head, int length, int direction, int keycol);
681     void sortLectureCustom(Lecture **head, int length, int direction, int keycol);
682     double compareStudent(Student *a, Student *b, int direction, int keycol);
683     double compareLecture(Lecture *a, Lecture *b, int direction, int keycol);
684
685     // 打印函数
686     void printStudent();
687     void printStudent(int studentNo);
688     void printStudent(const std::string &name);
689     void printLecture();
690     void printLecture(LectureType type);
691     void printLecture(const std::string &name);
692
693     // 百分制成绩转化为绩点
694     double calculateGPA(int score);
695
696     // 在修改后更新学生与课程
697     void updateStudent();
698     void updateLecture();
699
700     // 登录与注册函数
701     Account *login(std::string username, std::string password);
702     Account *registerUser(std::string username, std::string password, int permission);
703     void loadAccount(const std::string &userFilename, const std::string
&adminFilename);
704     void saveAccount(const std::string &userFilename, const std::string
&adminFilename);
705     int queryAccount(const std::string &username);
706 };
707
708 // 用户界面类
709 class UserInterface
710 {
711 private:
712     Database *database;           // 数据库指针
713     static Account *currentUser; // 当前用户指针

```

```
714
715 public:
716     // 构造函数
717     UserInterface();
718
719     // 析构函数
720     ~UserInterface();
721
722     // 运行与交互主函数
723     bool run();
724
725     // 功能函数
726     Account *login(); // 登录界面
727     void welcome();   // 欢迎界面
728     bool searchInfo();
729     bool searchStudent(); // 查询学生信息
730     bool searchLecture(); // 查询课程信息
731     bool sortInfo();
732     bool sortStudent(); // 排序学生信息
733     bool sortLecture(); // 排序课程信息
734     bool addInfo();
735     bool addStudent(); // 添加学生信息
736     bool addLecture(); // 添加课程信息
737     bool deleteInfo();
738     bool deleteStudent(); // 删除学生信息
739     bool deleteLecture(); // 删除课程信息
740     bool modifyInfo();
741     bool modifyStudent(); // 修改学生信息
742     bool modifyLecture(); // 修改课程信息
743     bool load();          // 读取文件信息
744     bool save();          // 保存文件信息
745     bool print();         // 打印信息
746     bool printStudent();  // 打印学生信息
747     bool printLecture();  // 打印课程信息
748     bool debug();         // 调试模式
749     bool about();         // 关于程序
750     void quit();          // 退出程序
751     void pause();         // 暂停程序
752 };
753
```

## 源文件

### account.cpp

```
1  #include "commonheader.h"
2
3  // 构造函数
4  Account::Account()
5  {
6      name = "DefaultName";
7      uid = currentUid++;
8  }
9
10 // 构造函数
11 Account::Account(std::string inputName, std::string inputPassword)
12 {
13     name = inputName;
14     password = inputPassword;
15     uid = currentUid++;
16 }
17
18 // 析构函数
19 Account::~Account() {}
20
21 // 获取密码
22 std::string Account::getPassword()
23 {
24     return password;
25 }
26
27 // 设置用户名
28 void Account::setName(std::string inputName)
29 {
30     name = inputName;
31 }
32
33 // 设置密码
34 void Account::setPassword(std::string inputPassword)
35 {
36     password = inputPassword;
37 }
38
39 // 重载">>"运算符
40 std::istream &operator>>(std::istream &is, Account &account)
41 {
42     is >> account.name >> account.password;
43     return is;
44 }
45
46 // 重载"<<"运算符
47 std::ostream &operator<<(std::ostream &os, Account &account)
```

```

48 {
49     os << account.name << "\t"
50     << account.password << std::endl;
51     return os;
52 }
53

```

## database.cpp

```

1  #include "commonheader.h"
2
3  // 构造函数
4  Database::Database()
5  {
6      studentList.makeEmpty();
7      requiredList.makeEmpty();
8      limitedList.makeEmpty();
9      optionalList.makeEmpty();
10     userList.makeEmpty();
11     adminList.makeEmpty();
12 }
13
14 // 析构函数
15 Database::~Database()
16 {
17     studentList.makeEmpty();
18     requiredList.makeEmpty();
19     limitedList.makeEmpty();
20     optionalList.makeEmpty();
21     userList.makeEmpty();
22     adminList.makeEmpty();
23 }
24
25 // 从文件中读取
26 void Database::load()
27 {
28     studentList.makeEmpty();
29     requiredList.makeEmpty();
30     limitedList.makeEmpty();
31     optionalList.makeEmpty();
32     try
33     {
34         loadStudent("savedata_student.dat");
35         loadRequired("savedata_required.dat");
36         loadLimited("savedata_limited.dat");
37         loadOptional("savedata_optional.dat");
38     }
39     catch (FileNotFoundException e)
40     {
41         throw e;
42     }

```

```
43 }
44
45 // 向文件中保存
46 void Database::save()
47 {
48     try
49     {
50         saveStudent("savedata_student.dat");
51         saveRequired("savedata_required.dat");
52         saveLimited("savedata_limited.dat");
53         saveOptional("savedata_optional.dat");
54     }
55     catch (FileNotFoundException e)
56     {
57         throw e;
58     }
59 }
60
61 // 获取学生列表长度
62 int Database::getStudentListSize()
63 {
64     return studentList.size();
65 }
66
67 // 获取必修课列表长度
68 int Database::getRequiredListSize()
69 {
70     return requiredList.size();
71 }
72
73 // 获取限选课列表长度
74 int Database::getLimitedListSize()
75 {
76     return limitedList.size();
77 }
78
79 // 获取任选课列表长度
80 int Database::getOptionalListSize()
81 {
82     return optionalList.size();
83 }
84
85 // 从文件中读取学生信息
86 void Database::loadStudent(const std::string &filename)
87 {
88     encrypt(filename);
89     std::ifstream in(filename.c_str(), std::ios::in);
90     Node<Student> *stu;
91     // 打开文件成功
92     if (in)
93     {
94         in.seekg(0, std::ios::end);
```

```

95     int fileSize = in.tellg();
96     in.seekg(std::ios::beg);
97     while (fileSize - in.tellg() > 2)
98     {
99         stu = new Node<Student>;
100         if (in >> stu->data)
101         {
102             studentList.insert_end(stu);
103         }
104         else
105         {
106             FileException e(filename, "operate", "read");
107             throw e;
108         }
109     }
110 }
111 else
112 {
113     FileException e(filename, "open", "read");
114     throw e;
115 }
116 in.close();
117 encrypt(filename);
118 }
119
120 // 从文件中读取必修课信息
121 void Database::loadRequired(const std::string &filename)
122 {
123     encrypt(filename);
124     std::ifstream in(filename.c_str(), std::ios::in);
125     Node<LectureRequired> *req;
126     // 打开文件成功
127     if (in)
128     {
129         in.seekg(0, std::ios::end);
130         int fileSize = in.tellg();
131         in.seekg(std::ios::beg);
132         while (fileSize - in.tellg() > 2)
133         {
134             req = new Node<LectureRequired>;
135             if (in >> req->data)
136             {
137                 requiredList.insert_end(req);
138             }
139             else
140             {
141                 FileException e(filename, "operate", "read");
142                 throw e;
143             }
144         }
145     }
146     else

```

```

147     {
148         FileException e(filename, "open", "read");
149         throw e;
150     }
151     in.close();
152     encrypt(filename);
153 }
154
155 // 从文件中读取限选课信息
156 void Database::loadLimited(const std::string &filename)
157 {
158     encrypt(filename);
159     std::ifstream in(filename.c_str(), std::ios::in);
160     Node<LectureLimited> *lim;
161     // 打开文件成功
162     if (in)
163     {
164         in.seekg(0, std::ios::end);
165         int fileSize = in.tellg();
166         in.seekg(std::ios::beg);
167         while (fileSize - in.tellg() > 2)
168         {
169             lim = new Node<LectureLimited>;
170             if (in >> lim->data)
171             {
172                 limitedList.insert_end(lim);
173             }
174             else
175             {
176                 FileException e(filename, "operate", "read");
177                 throw e;
178             }
179         }
180     }
181     else
182     {
183         FileException e(filename, "open", "read");
184         throw e;
185     }
186     in.close();
187     encrypt(filename);
188 }
189
190 // 从文件中读取任选课信息
191 void Database::loadOptional(const std::string &filename)
192 {
193     encrypt(filename);
194     std::ifstream in(filename.c_str(), std::ios::in);
195     Node<LectureOptional> *opt;
196     // 打开文件成功
197     if (in)
198     {

```

```

199     in.seekg(0, std::ios::end);
200     int fileSize = in.tellg();
201     in.seekg(std::ios::beg);
202     while (fileSize - in.tellg() > 2)
203     {
204         opt = new Node<LectureOptional>;
205         if (in >> opt->data)
206         {
207             optionalList.insert_end(opt);
208         }
209         else
210         {
211             FileException e(filename, "operate", "read");
212             throw e;
213         }
214     }
215 }
216 else
217 {
218     FileException e(filename, "open", "read");
219     throw e;
220 }
221 in.close();
222 encrypt(filename);
223 }
224
225 // 向文件中保存学生信息
226 void Database::saveStudent(const std::string &filename)
227 {
228     encrypt(filename);
229     std::ofstream out(filename.c_str(), std::ios::out | std::ios::trunc);
230     Node<Student> *stu;
231     // 打开文件成功
232     if (out)
233     {
234         stu = studentList.getFirst();
235         while (stu != nullptr)
236         {
237             if (!(out << stu->data))
238             {
239                 FileException e(filename, "operate", "write");
240                 throw e;
241             }
242             stu = stu->next;
243         }
244     }
245     else
246     {
247         FileException e(filename, "open", "write");
248         throw e;
249     }
250     out.close();

```



```

251     encrypt(filename);
252 }
253
254 // 向文件中保存必修课信息
255 void Database::saveRequired(const std::string &filename)
256 {
257     encrypt(filename);
258     std::ofstream out(filename.c_str(), std::ios::out | std::ios::trunc);
259     Node<LectureRequired> *req;
260     // 打开文件成功
261     if (out)
262     {
263         req = requiredList.getFirst();
264         while (req != nullptr)
265         {
266             if (!(out << req->data))
267             {
268                 FileException e(filename, "operate", "write");
269                 throw e;
270             }
271             req = req->next;
272         }
273     }
274     else
275     {
276         FileException e(filename, "open", "write");
277         throw e;
278     }
279     out.close();
280     encrypt(filename);
281 }
282
283 // 向文件中保存限选课信息
284 void Database::saveLimited(const std::string &filename)
285 {
286     encrypt(filename);
287     std::ofstream out(filename.c_str(), std::ios::out | std::ios::trunc);
288     Node<LectureLimited> *lim;
289     // 打开文件成功
290     if (out)
291     {
292         lim = limitedList.getFirst();
293         while (lim != nullptr)
294         {
295             if (!(out << lim->data))
296             {
297                 FileException e(filename, "operate", "write");
298                 throw e;
299             }
300             lim = lim->next;
301         }
302     }

```

```

303     else
304     {
305         FileException e(filename, "open", "write");
306         throw e;
307     }
308     out.close();
309     encrypt(filename);
310 }
311
312 // 向文件中保存任选课信息
313 void Database::saveOptional(const std::string &filename)
314 {
315     encrypt(filename);
316     std::ofstream out(filename.c_str(), std::ios::out | std::ios::trunc);
317     Node<LectureOptional> *opt;
318     // 打开文件成功
319     if (out)
320     {
321         opt = optionallist.getFirst();
322         while (opt != nullptr)
323         {
324             if (!(out << opt->data))
325             {
326                 FileException e(filename, "operate", "write");
327                 throw e;
328             }
329             opt = opt->next;
330         }
331     }
332     else
333     {
334         FileException e(filename, "open", "write");
335         throw e;
336     }
337     out.close();
338     encrypt(filename);
339 }
340
341 // 加密文件
342 void Database::encrypt(const std::string &filename)
343 {
344     std::vector<std::string> codestr;
345     std::ifstream in(filename.c_str(), std::ios::in);
346     std::string tmp;
347     while (std::getline(in, tmp))
348     {
349         codestr.push_back(tmp);
350     }
351     in.close();
352     std::ofstream out(filename.c_str(), std::ios::out | std::ios::trunc);
353     for (int i = 0; i < codestr.size(); i++)
354     {

```

```

355     for (int j = 0; j < codestr[i].size(); j++)
356     {
357         if ((char)~codestr[i][j] != 0x1a)
358         {
359             out << (char)~codestr[i][j];
360         }
361         else
362         {
363             out << (char)codestr[i][j];
364         }
365     }
366     out << std::endl;
367 }
368 out.close();
369 }
370
371 // 更安全的加密文件
372 void Database::encrypt_key(const std::string &filename, const std::string
&keyFilename)
373 {
374     std::string key;
375     std::ifstream key_in(keyFilename.c_str(), std::ios::in);
376     std::getline(key_in, key);
377     key_in.close();
378     std::vector<std::string> codestr;
379     std::ifstream in(filename.c_str(), std::ios::in);
380     std::string tmp;
381     while (std::getline(in, tmp))
382     {
383         codestr.push_back(tmp);
384     }
385     in.close();
386     std::ofstream out(filename.c_str(), std::ios::out | std::ios::trunc);
387     for (int i = 0; i < codestr.size(); i++)
388     {
389         for (int j = 0; j < codestr[i].size(); j++)
390         {
391             out << (codestr[i][j] ^ key[(i + j) % key.length()]);
392         }
393         out << std::endl;
394     }
395     out.close();
396 }
397
398 // 生成密钥
399 void Database::key_gen(const std::string &filename)
400 {
401     std::string key = std::to_string(time(0));
402     std::ofstream out(filename.c_str(), std::ios::out | std::ios::trunc);
403     out << key;
404     out.close();
405 }

```

```

406
407 // 时间戳转换
408 std::string Database::timeStampToString(const time_t &timeStamp)
409 {
410     struct tm *timeinfo;
411     char buffer[80];
412     timeinfo = localtime(&timeStamp);
413     strftime(buffer, 80, "%Y-%m-%d %H:%M:%S", timeinfo);
414     return std::string(buffer);
415 }
416
417 // 以姓名为准删除学生
418 bool Database::deleteStudent(const std::string &name)
419 {
420     Node<Student> *stu = studentList.getFirst();
421     bool flag = false;
422     if (stu != nullptr)
423     {
424         do
425         {
426             if (stu->data.getName() == name)
427             {
428                 stu->data.printStudentInfo();
429                 std::cout << "是否确认删除? [Y/N]" << std::endl;
430                 char input;
431                 std::cin >> input;
432                 if (input == 'Y' || input == 'y')
433                 {
434                     studentList.remove(stu);
435                     flag = true;
436                     std::cout << "删除成功!" << std::endl;
437                 }
438                 else
439                 {
440                     std::cout << "取消删除!" << std::endl;
441                 }
442             }
443             stu = studentList.getNext();
444         } while (stu != nullptr);
445     }
446     return flag;
447 }
448
449 // 以学号为准删除学生
450 bool Database::deleteStudent(int studentNo)
451 {
452     Node<Student> *stu = studentList.getFirst();
453     bool flag = false;
454     if (stu != nullptr)
455     {
456         do
457         {

```

```

458         if (stu->data.getStudentNo() == studentNo)
459         {
460             stu->data.printStudentInfo();
461             std::cout << "是否确认删除? [Y/N]" << std::endl;
462             char input;
463             input = _getch();
464             if (input == 'Y' || input == 'y')
465             {
466                 studentList.remove(stu);
467                 flag = true;
468                 std::cout << "删除成功! " << std::endl;
469             }
470             else
471             {
472                 std::cout << "取消删除! " << std::endl;
473             }
474         }
475         stu = studentList.getNext();
476     } while (stu != nullptr);
477 }
478 return flag;
479 }
480
481 // 以课程名称为准删除必修课
482 bool Database::deleteRequired(const std::string &name)
483 {
484     Node<LectureRequired> *req = requiredList.getFirst();
485     bool flag = false;
486     if (req != nullptr)
487     {
488         do
489         {
490             if (req->data.getName() == name)
491             {
492                 req->data.printLectureInfo();
493                 std::cout << "是否确认删除? [Y/N]" << std::endl;
494                 char input;
495                 std::cin >> input;
496                 if (input == 'Y' || input == 'y')
497                 {
498                     requiredList.remove(req);
499                     flag = true;
500                     std::cout << "删除成功! " << std::endl;
501                 }
502                 else
503                 {
504                     std::cout << "取消删除! " << std::endl;
505                 }
506             }
507             req = requiredList.getNext();
508         } while (req != nullptr);
509     }

```

```

510     return flag;
511 }
512
513 // 以课程名称为准删除限选课
514 bool Database::deleteLimited(const std::string &name)
515 {
516     Node<LectureLimited> *lim = limitedList.getFirst();
517     bool flag = false;
518     if (lim != nullptr)
519     {
520         do
521         {
522             if (lim->data.getName() == name)
523             {
524                 lim->data.printLectureInfo();
525                 std::cout << "是否确认删除? [Y/N]" << std::endl;
526                 char input;
527                 std::cin >> input;
528                 if (input == 'Y' || input == 'y')
529                 {
530                     limitedList.remove(lim);
531                     flag = true;
532                     std::cout << "删除成功! " << std::endl;
533                 }
534                 else
535                 {
536                     std::cout << "取消删除! " << std::endl;
537                 }
538             }
539             lim = limitedList.getNext();
540         } while (lim != nullptr);
541     }
542     return flag;
543 }
544
545 // 以课程名称为准删除任选课
546 bool Database::deleteOptional(const std::string &name)
547 {
548     Node<LectureOptional> *opt = optionalList.getFirst();
549     bool flag = false;
550     if (opt != nullptr)
551     {
552         do
553         {
554             if (opt->data.getName() == name)
555             {
556                 opt->data.printLectureInfo();
557                 std::cout << "是否确认删除? [Y/N]" << std::endl;
558                 char input;
559                 std::cin >> input;
560                 if (input == 'Y' || input == 'y')
561                 {

```

```

562         optionalList.remove(opt);
563         flag = true;
564         std::cout << "删除成功! " << std::endl;
565     }
566     else
567     {
568         std::cout << "取消删除! " << std::endl;
569     }
570 }
571 opt = optionalList.getNext();
572 } while (opt != nullptr);
573 }
574 return flag;
575 }
576
577 // 新增学生
578 void Database::addStudent()
579 {
580     Node<Student> *stu = new Node<Student>;
581     stu->data.updateInfo(*this);
582     studentList.insert_end(stu);
583 }
584
585 // 新增必修课
586 void Database::addRequired(const std::string &name, int credit)
587 {
588     Node<LectureRequired> *req = new Node<LectureRequired>;
589     req->data.setName(name);
590     req->data.setCredit(credit);
591     requiredList.insert_end(req);
592 }
593
594 // 新增限选课
595 void Database::addLimited(const std::string &name, int credit)
596 {
597     Node<LectureLimited> *lim = new Node<LectureLimited>;
598     lim->data.setName(name);
599     lim->data.setCredit(credit);
600     limitedList.insert_end(lim);
601 }
602
603 // 新增任选课
604 void Database::addOptional(const std::string &name, int credit)
605 {
606     Node<LectureOptional> *opt = new Node<LectureOptional>;
607     opt->data.setName(name);
608     opt->data.setCredit(credit);
609     optionalList.insert_end(opt);
610 }
611
612 // 将学生信息添加到课程

```

```

613 void Database::addStudentToLecture(const std::string &name, LectureType type, Student
    &stu)
614 {
615     int dest = 0;
616     for (int i = 0; i < stu.getLectureName().size(); i++)
617     {
618         if (stu.getLectureName()[i] == name)
619         {
620             dest = i;
621             break;
622         }
623     }
624     switch (type)
625     {
626     case REQUIRED:
627     {
628         Node<LectureRequired> *req = requiredList.getFirst();
629         if (req != nullptr)
630         {
631             do
632             {
633                 if (req->data.getName() == name)
634                 {
635                     req->data.addStudent(stu.getStudentNo(), stu.getName(),
stu.getLectureScore()[dest], stu.getLectureGPA()[dest]);
636                 }
637                 req = requiredList.getNext();
638             } while (req != nullptr);
639         }
640         break;
641     }
642     case LIMITED:
643     {
644         Node<LectureLimited> *lim = limitedList.getFirst();
645         if (lim != nullptr)
646         {
647             do
648             {
649                 if (lim->data.getName() == name)
650                 {
651                     lim->data.addStudent(stu.getStudentNo(), stu.getName(),
stu.getLectureScore()[dest], stu.getLectureGPA()[dest]);
652                 }
653                 lim = limitedList.getNext();
654             } while (lim != nullptr);
655         }
656         break;
657     }
658     case OPTIONAL:
659     {
660         Node<LectureOptional> *opt = optionalList.getFirst();
661         if (opt != nullptr)

```



```

662     {
663         do
664         {
665             if (opt->data.getName() == name)
666             {
667                 opt->data.addStudent(stu.getStudentNo(), stu.getName(),
stu.getLectureScore()[dest], stu.getLectureGPA()[dest], stu.getLecturePF()[dest]);
668             }
669             opt = optionalList.getNext();
670         } while (opt != nullptr);
671     }
672     break;
673 }
674 default:
675     break;
676 }
677 }
678
679 ////输出学生信息
680 // void Database::displayStudent(const std::string& name)
681 //{
682 // Node<Student>* stu = studentList.getFirst();
683 // if (stu != nullptr)
684 // {
685 //     do
686 //     {
687 //         if (stu->data.getName() == name)
688 //         {
689 //             stu->data.printStudentInfo();
690 //         }
691 //         stu = studentList.getNext();
692 //     } while (stu != nullptr);
693 // }
694 // }
695 //
696 ////输出学生信息
697 // void Database::displayStudent(int studentNo)
698 //{
699 // Node<Student>* stu = studentList.getFirst();
700 // if (stu != nullptr)
701 // {
702 //     do
703 //     {
704 //         if (stu->data.getStudentNo() == studentNo)
705 //         {
706 //             stu->data.printStudentInfo();
707 //         }
708 //         stu = studentList.getNext();
709 //     } while (stu != nullptr);
710 // }
711 // }
712 //

```

```

713  ////输出课程信息
714  // void Database::displayLecture(const std::string& name)
715  //{
716  //  displayRequired(name);
717  //  displayLimited(name);
718  //  displayOptional(name);
719  // }
720  //
721  ////输出必修课信息
722  // void Database::displayRequired(const std::string& name)
723  //{
724  //  Node<LectureRequired>* req = requiredList.getFirst();
725  //  if (req != nullptr)
726  //  {
727  //      do
728  //      {
729  //          if (req->data.getName() == name)
730  //          {
731  //              req->data.printLectureInfo();
732  //          }
733  //          req = requiredList.getNext();
734  //      } while (req != nullptr);
735  //  }
736  // }
737  //
738  ////输出限选课信息
739  // void Database::displayLimited(const std::string& name)
740  //{
741  //  Node<LectureLimited>* lim = limitedList.getFirst();
742  //  if (lim != nullptr)
743  //  {
744  //      do
745  //      {
746  //          if (lim->data.getName() == name)
747  //          {
748  //              lim->data.printLectureInfo();
749  //          }
750  //          lim = limitedList.getNext();
751  //      } while (lim != nullptr);
752  //  }
753  // }
754  //
755  ////输出任选课信息
756  // void Database::displayOptional(const std::string& name)
757  //{
758  //  Node<LectureOptional>* opt = optionalList.getFirst();
759  //  if (opt != nullptr)
760  //  {
761  //      do
762  //      {
763  //          if (opt->data.getName() == name)
764  //          {

```

```

765 //          opt->data.printLectureInfo();
766 //      }
767 //          opt = optionalList.getNext();
768 //      } while (opt != nullptr);
769 //  }
770 // }
771 //
772 ////输出所有学生
773 // int Database::displayAllStudent()
774 //{
775 //  Node<Student>* stu = studentList.getFirst();
776 //  int count = 0;
777 //  if (stu != nullptr)
778 //  {
779 //      do
780 //      {
781 //          stu->data.printStudentInfo();
782 //          count++;
783 //          stu = studentList.getNext();
784 //      } while (stu != nullptr);
785 //  }
786 //  return count;
787 // }
788 //
789 ////输出所有课程
790 // int Database::displayAllLecture(LectureType type)
791 //{
792 //  Node<LectureRequired>* req = requiredList.getFirst();
793 //  Node<LectureLimited>* lim = limitedList.getFirst();
794 //  Node<LectureOptional>* opt = optionalList.getFirst();
795 //  int count = 0;
796 //  switch (type)
797 //  {
798 //      case DEFAULT:
799 //          if (req != nullptr)
800 //          {
801 //              do
802 //              {
803 //                  req->data.printLectureInfo();
804 //                  count++;
805 //                  req = requiredList.getNext();
806 //              } while (req != nullptr);
807 //          }
808 //          if (lim != nullptr)
809 //          {
810 //              do
811 //              {
812 //                  lim->data.printLectureInfo();
813 //                  count++;
814 //                  lim = limitedList.getNext();
815 //              } while (lim != nullptr);
816 //          }

```

```

817 //     if (opt != nullptr)
818 //     {
819 //         do
820 //         {
821 //             opt->data.printLectureInfo();
822 //             count++;
823 //             opt = optionalList.getNext();
824 //         } while (opt != nullptr);
825 //     }
826 //     break;
827 // case REQUIRED:
828 //     if (req != nullptr)
829 //     {
830 //         do
831 //         {
832 //             req->data.printLectureInfo();
833 //             count++;
834 //             req = requiredList.getNext();
835 //         } while (req != nullptr);
836 //     }
837 //     break;
838 // case LIMITED:
839 //     if (lim != nullptr)
840 //     {
841 //         do
842 //         {
843 //             lim->data.printLectureInfo();
844 //             count++;
845 //             lim = limitedList.getNext();
846 //         } while (lim != nullptr);
847 //     }
848 //     break;
849 // case OPTIONAL:
850 //     if (opt != nullptr)
851 //     {
852 //         do
853 //         {
854 //             opt->data.printLectureInfo();
855 //             count++;
856 //             opt = optionalList.getNext();
857 //         } while (opt != nullptr);
858 //     }
859 //     break;
860 // default:
861 //     break;
862 // }
863 // return count;
864 // }
865
866 // 按名称修改学生信息
867 bool Database::modifyStudent(const std::string &name)
868 {

```

```

869     Node<Student> *stu = studentList.getFirst();
870     bool flag = false;
871     if (stu != nullptr)
872     {
873         do
874         {
875             if (stu->data.getName() == name)
876             {
877                 flag = true;
878                 stu->data.printStudentInfo();
879                 std::cout << "是否确认修改? [Y/N]" << std::endl;
880                 char input;
881                 std::cin >> input;
882                 if (input == 'Y' || input == 'y')
883                 {
884                     stu->data.updateInfo(*this);
885                     std::cout << "修改成功! " << std::endl;
886                 }
887                 else
888                 {
889                     std::cout << "取消修改! " << std::endl;
890                 }
891             }
892             stu = studentList.getNext();
893         } while (stu != nullptr);
894     }
895     return flag;
896 }
897
898 // 按学号修改学生信息
899 bool Database::modifyStudent(int studentNo)
900 {
901     Node<Student> *stu = studentList.getFirst();
902     bool flag = false;
903     if (stu != nullptr)
904     {
905         do
906         {
907             if (stu->data.getStudentNo() == studentNo)
908             {
909                 flag = true;
910                 stu->data.printStudentInfo();
911                 std::cout << "是否确认修改? [Y/N]" << std::endl;
912                 char input;
913                 std::cin >> input;
914                 if (input == 'Y' || input == 'y')
915                 {
916                     stu->data.updateInfo(*this);
917                     std::cout << "修改成功! " << std::endl;
918                 }
919                 else
920                 {

```

```

921         std::cout << "取消修改! " << std::endl;
922     }
923 }
924     stu = studentList.getNext();
925 } while (stu != nullptr);
926 }
927     return flag;
928 }
929
930 // 按名称修改课程信息
931 bool Database::modifyLecture(const std::string &name)
932 {
933     bool flag = false;
934     flag = modifyRequired(name) + modifyLimited(name) + modifyOptional(name);
935     return flag;
936 }
937
938 // 按名称修改必修课信息
939 bool Database::modifyRequired(const std::string &name)
940 {
941     Node<LectureRequired> *req = requiredList.getFirst();
942     bool flag = false;
943     if (req != nullptr)
944     {
945         do
946         {
947             if (req->data.getName() == name)
948             {
949                 flag = true;
950                 req->data.printLectureInfo();
951                 std::cout << "是否确认修改? [Y/N]" << std::endl;
952                 char input;
953                 input = _getch();
954                 if (input == 'Y' || input == 'y')
955                 {
956                     req->data.updateInfo(*this);
957                     std::cout << "修改成功! " << std::endl;
958                 }
959                 else
960                 {
961                     std::cout << "取消修改! " << std::endl;
962                 }
963             }
964             req = requiredList.getNext();
965         } while (req != nullptr);
966     }
967     return flag;
968 }
969
970 // 按名称修改限选课信息
971 bool Database::modifyLimited(const std::string &name)
972 {

```

```

973     Node<LectureLimited> *lim = limitedList.getFirst();
974     bool flag = false;
975     if (lim != nullptr)
976     {
977         do
978         {
979             if (lim->data.getName() == name)
980             {
981                 flag = true;
982                 lim->data.printLectureInfo();
983                 std::cout << "是否确认修改? [Y/N]" << std::endl;
984                 char input;
985                 input = _getch();
986                 if (input == 'Y' || input == 'y')
987                 {
988                     lim->data.updateInfo(*this);
989                     std::cout << "修改成功! " << std::endl;
990                 }
991                 else
992                 {
993                     std::cout << "取消修改! " << std::endl;
994                 }
995             }
996             lim = limitedList.getNext();
997         } while (lim != nullptr);
998     }
999     return flag;
1000 }
1001
1002 // 按名称修改任选课信息
1003 bool Database::modifyOptional(const std::string &name)
1004 {
1005     Node<LectureOptional> *opt = optionalList.getFirst();
1006     bool flag = false;
1007     if (opt != nullptr)
1008     {
1009         do
1010         {
1011             if (opt->data.getName() == name)
1012             {
1013                 flag = true;
1014                 opt->data.printLectureInfo();
1015                 std::cout << "是否确认修改? [Y/N]" << std::endl;
1016                 char input;
1017                 input = _getch();
1018                 if (input == 'Y' || input == 'y')
1019                 {
1020                     opt->data.updateInfo(*this);
1021                     std::cout << "修改成功! " << std::endl;
1022                 }
1023                 else
1024                 {

```

```

1025         std::cout << "取消修改!" << std::endl;
1026     }
1027 }
1028     opt = optionalList.getNext();
1029 } while (opt != nullptr);
1030 }
1031 return flag;
1032 }
1033
1034 // 按名称查询学生信息
1035 int Database::queryStudent(const std::string &name, bool display)
1036 {
1037     Node<Student> *stu = studentList.getFirst();
1038     int count = 0;
1039     if (stu != nullptr)
1040     {
1041         do
1042         {
1043             if (stu->data.getName() == name)
1044             {
1045                 if (display)
1046                     stu->data.printStudentInfo();
1047                 count++;
1048             }
1049             stu = studentList.getNext();
1050         } while (stu != nullptr);
1051     }
1052     return count;
1053 }
1054
1055 // 按学号查询学生信息
1056 int Database::queryStudent(int studentNo, bool display)
1057 {
1058     Node<Student> *stu = studentList.getFirst();
1059     int count = 0;
1060     if (stu != nullptr)
1061     {
1062         do
1063         {
1064             if (stu->data.getStudentNo() == studentNo)
1065             {
1066                 if (display)
1067                     stu->data.printStudentInfo();
1068                 count++;
1069             }
1070             stu = studentList.getNext();
1071         } while (stu != nullptr);
1072     }
1073     return count;
1074 }
1075
1076 // 按名称查询课程信息

```



```

1077 int Database::queryLecture(const std::string &name, bool display)
1078 {
1079     int count = 0;
1080     count = queryRequired(name, display) + queryLimited(name, display) +
queryOptional(name, display);
1081     return count;
1082 }
1083
1084 // 按名称查询必修课信息
1085 int Database::queryRequired(const std::string &name, bool display)
1086 {
1087     Node<LectureRequired> *req = requiredList.getFirst();
1088     int count = 0;
1089     if (req != nullptr)
1090     {
1091         do
1092         {
1093             if (req->data.getName() == name)
1094             {
1095                 if (display)
1096                     req->data.printLectureInfo();
1097                 count++;
1098             }
1099             req = requiredList.getNext();
1100         } while (req != nullptr);
1101     }
1102     return count;
1103 }
1104
1105 // 按名称查询限选课信息
1106 int Database::queryLimited(const std::string &name, bool display)
1107 {
1108     Node<LectureLimited> *lim = limitedList.getFirst();
1109     int count = 0;
1110     if (lim != nullptr)
1111     {
1112         do
1113         {
1114             if (lim->data.getName() == name)
1115             {
1116                 if (display)
1117                     lim->data.printLectureInfo();
1118                 count++;
1119             }
1120             lim = limitedList.getNext();
1121         } while (lim != nullptr);
1122     }
1123     return count;
1124 }
1125
1126 // 按名称查询任选课信息
1127 int Database::queryOptional(const std::string &name, bool display)

```

```

1128 {
1129     Node<LectureOptional> *opt = optionalList.getFirst();
1130     int count = 0;
1131     if (opt != nullptr)
1132     {
1133         do
1134         {
1135             if (opt->data.getName() == name)
1136             {
1137                 if (display)
1138                     opt->data.printLectureInfo();
1139                 count++;
1140             }
1141             opt = optionalList.getNext();
1142         } while (opt != nullptr);
1143     }
1144     return count;
1145 }
1146
1147 // 按名称找到学生信息
1148 Student *Database::findStudent(const std::string &name)
1149 {
1150     Node<Student> *stu = studentList.getFirst();
1151     if (stu != nullptr)
1152     {
1153         do
1154         {
1155             if (stu->data.getName() == name)
1156             {
1157                 return &stu->data;
1158             }
1159             stu = studentList.getNext();
1160         } while (stu != nullptr);
1161     }
1162     return nullptr;
1163 }
1164
1165 // 按学号找到学生信息
1166 Student *Database::findStudent(int studentNo)
1167 {
1168     Node<Student> *stu = studentList.getFirst();
1169     if (stu != nullptr)
1170     {
1171         do
1172         {
1173             if (stu->data.getStudentNo() == studentNo)
1174             {
1175                 return &stu->data;
1176             }
1177             stu = studentList.getNext();
1178         } while (stu != nullptr);
1179     }

```

```

1180     return nullptr;
1181 }
1182
1183 // 按名称找到课程信息
1184 Lecture *Database::findLecture(const std::string &name)
1185 {
1186     Lecture *lec = findRequired(name);
1187     if (lec != nullptr)
1188         return lec;
1189     lec = findLimited(name);
1190     if (lec != nullptr)
1191         return lec;
1192     lec = findOptional(name);
1193     if (lec != nullptr)
1194         return lec;
1195     return nullptr;
1196 }
1197
1198 // 按名称找到必修课信息
1199 LectureRequired *Database::findRequired(const std::string &name)
1200 {
1201     Node<LectureRequired> *req = requiredList.getFirst();
1202     if (req != nullptr)
1203     {
1204         do
1205         {
1206             if (req->data.getName() == name)
1207             {
1208                 return &req->data;
1209             }
1210             req = requiredList.getNext();
1211         } while (req != nullptr);
1212     }
1213     return nullptr;
1214 }
1215
1216 // 按名称找到限选课信息
1217 LectureLimited *Database::findLimited(const std::string &name)
1218 {
1219     Node<LectureLimited> *lim = limitedList.getFirst();
1220     if (lim != nullptr)
1221     {
1222         do
1223         {
1224             if (lim->data.getName() == name)
1225             {
1226                 return &lim->data;
1227             }
1228             lim = limitedList.getNext();
1229         } while (lim != nullptr);
1230     }
1231     return nullptr;

```

```

1232 }
1233
1234 // 按名称找到任选课信息
1235 LectureOptional *Database::findOptional(const std::string &name)
1236 {
1237     Node<LectureOptional> *opt = optionalList.getFirst();
1238     if (opt != nullptr)
1239     {
1240         do
1241         {
1242             if (opt->data.getName() == name)
1243             {
1244                 return &opt->data;
1245             }
1246             opt = optionalList.getNext();
1247         } while (opt != nullptr);
1248     }
1249     return nullptr;
1250 }
1251
1252 // 按关键列排序学生, direction为1表示升序, 为-1表示降序, keycol为关键列
1253 // 关键列: 0-studentNo, 1-name, 2-totalScore, 3-totalCredit, 4-totalGPA, 5-
averageScore, 6-averageGPA
1254 void Database::sortStudent(int direction, int keycol)
1255 {
1256     Student **head;
1257     int length = studentList.size();
1258     if (length > 0)
1259     {
1260         head = new Student *[length];
1261         Node<Student> *tmp = studentList.getFirst();
1262         int cnt = 0;
1263         do
1264         {
1265             head[cnt] = &tmp->data;
1266             tmp = studentList.getNext();
1267             cnt++;
1268         } while (tmp != nullptr);
1269         sortStudentCustom(head, length, direction, keycol);
1270         int widthStudentNo = 5, widthName = 5;
1271         for (cnt = 0; cnt < length; cnt++)
1272         {
1273             if (std::to_string(head[cnt]->getStudentNo()).length() > widthStudentNo)
1274                 widthStudentNo = std::to_string(head[cnt]->getStudentNo()).length() +
1;
1275             if (head[cnt]->getName().length() > widthName)
1276                 widthName = head[cnt]->getName().length() + 1;
1277         }
1278         std::cout.width(widthStudentNo);
1279         std::cout << "学号";
1280         std::cout.width(widthName);
1281         std::cout << "姓名";

```

```

1282         std::cout.width(5);
1283         std::cout << "课数";
1284         std::cout.width(5);
1285         std::cout << "学分";
1286         std::cout.width(5);
1287         std::cout << "均分";
1288         std::cout.width(5);
1289         std::cout << "均绩" << std::endl;
1290         for (cnt = 0; cnt < length; cnt++)
1291             head[cnt]->printInfo(widthStudentNo, widthName);
1292         delete[] head;
1293     }
1294     else
1295     {
1296         std::cout << "学生信息为空! " << std::endl;
1297     }
1298 }
1299
1300 // 按关键列排序课程, direction为1表示升序, 为-1表示降序, keycol为关键列
1301 // 关键列: 0-name, 1-credit, 2-studentNum, 3-averageScore, 4-averageGPA
1302 void Database::sortLecture(LectureType type, int direction, int keycol)
1303 {
1304     Lecture **head;
1305     int length = 0;
1306     switch (type)
1307     {
1308     case DEFAULT:
1309         length = requiredList.size() + limitedList.size() + optionalList.size();
1310         if (length > 0)
1311         {
1312             head = new Lecture *[length];
1313             int cnt = 0;
1314             Node<LectureRequired> *req = requiredList.getFirst();
1315             if (req != nullptr)
1316             {
1317                 do
1318                 {
1319                     head[cnt] = &req->data;
1320                     req = requiredList.getNext();
1321                     cnt++;
1322                 } while (req != nullptr);
1323             }
1324             Node<LectureLimited> *lim = limitedList.getFirst();
1325             if (lim != nullptr)
1326             {
1327                 do
1328                 {
1329                     head[cnt] = &lim->data;
1330                     lim = limitedList.getNext();
1331                     cnt++;
1332                 } while (lim != nullptr);
1333             }

```

```

1334     Node<LectureOptional> *opt = optionalList.getFirst();
1335     if (opt != nullptr)
1336     {
1337         do
1338         {
1339             head[cnt] = &opt->data;
1340             opt = optionalList.getNext();
1341             cnt++;
1342         } while (opt != nullptr);
1343     }
1344     sortLectureCustom(head, length, direction, keycol);
1345     int widthName = 5;
1346     for (cnt = 0; cnt < length; cnt++)
1347     {
1348         if (head[cnt]->getName().length() > widthName)
1349             widthName = head[cnt]->getName().length() + 1;
1350     }
1351     std::cout.width(widthName);
1352     std::cout << "名称";
1353     std::cout.width(5);
1354     std::cout << "类型";
1355     std::cout.width(5);
1356     std::cout << "学分";
1357     std::cout.width(5);
1358     std::cout << "人数";
1359     std::cout.width(5);
1360     std::cout << "均分";
1361     std::cout.width(5);
1362     std::cout << "均绩" << std::endl;
1363     for (cnt = 0; cnt < length; cnt++)
1364         head[cnt]->printInfo(widthName);
1365     delete[] head;
1366 }
1367 else
1368 {
1369     std::cout << "课程信息为空!" << std::endl;
1370 }
1371 break;
1372 case REQUIRED:
1373     length = requiredList.size();
1374     if (length > 0)
1375     {
1376         head = new Lecture *[length];
1377         int cnt = 0;
1378         Node<LectureRequired> *req = requiredList.getFirst();
1379         if (req != nullptr)
1380         {
1381             do
1382             {
1383                 head[cnt] = &req->data;
1384                 req = requiredList.getNext();
1385                 cnt++;

```

```

1386         } while (req != nullptr);
1387     }
1388     sortLectureCustom(head, length, direction, keycol);
1389     int widthName = 5;
1390     for (cnt = 0; cnt < length; cnt++)
1391     {
1392         if (head[cnt]->getName().length() > widthName)
1393             widthName = head[cnt]->getName().length() + 1;
1394     }
1395     std::cout.width(widthName);
1396     std::cout << "名称";
1397     std::cout.width(5);
1398     std::cout << "类型";
1399     std::cout.width(5);
1400     std::cout << "学分";
1401     std::cout.width(5);
1402     std::cout << "人数";
1403     std::cout.width(5);
1404     std::cout << "均分";
1405     std::cout.width(5);
1406     std::cout << "均绩" << std::endl;
1407     for (cnt = 0; cnt < length; cnt++)
1408         head[cnt]->printInfo(widthName);
1409     delete[] head;
1410 }
1411 else
1412 {
1413     std::cout << "必修课信息为空！" << std::endl;
1414 }
1415 break;
1416 case LIMITED:
1417     length = limitedList.size();
1418     if (length > 0)
1419     {
1420         head = new Lecture *[length];
1421         int cnt = 0;
1422         Node<LectureLimited> *lim = limitedList.getFirst();
1423         if (lim != nullptr)
1424         {
1425             do
1426             {
1427                 head[cnt] = &lim->data;
1428                 lim = limitedList.getNext();
1429                 cnt++;
1430             } while (lim != nullptr);
1431         }
1432         sortLectureCustom(head, length, direction, keycol);
1433         int widthName = 5;
1434         for (cnt = 0; cnt < length; cnt++)
1435         {
1436             if (head[cnt]->getName().length() > widthName)
1437                 widthName = head[cnt]->getName().length() + 1;

```

```

1438     }
1439     std::cout.width(widthName);
1440     std::cout << "名称";
1441     std::cout.width(5);
1442     std::cout << "类型";
1443     std::cout.width(5);
1444     std::cout << "学分";
1445     std::cout.width(5);
1446     std::cout << "人数";
1447     std::cout.width(5);
1448     std::cout << "均分";
1449     std::cout.width(5);
1450     std::cout << "均绩" << std::endl;
1451     for (cnt = 0; cnt < length; cnt++)
1452         head[cnt]->printInfo(widthName);
1453     delete[] head;
1454 }
1455 else
1456 {
1457     std::cout << "限选课信息为空！" << std::endl;
1458 }
1459 break;
1460 case OPTIONAL:
1461     length = optionalList.size();
1462     if (length > 0)
1463     {
1464         head = new Lecture *[length];
1465         int cnt = 0;
1466         Node<LectureOptional> *opt = optionalList.getFirst();
1467         if (opt != nullptr)
1468         {
1469             do
1470             {
1471                 head[cnt] = &opt->data;
1472                 opt = optionalList.getNext();
1473                 cnt++;
1474             } while (opt != nullptr);
1475         }
1476         sortLectureCustom(head, length, direction, keycol);
1477         int widthName = 5;
1478         for (cnt = 0; cnt < length; cnt++)
1479         {
1480             if (head[cnt]->getName().length() > widthName)
1481                 widthName = head[cnt]->getName().length() + 1;
1482         }
1483         std::cout.width(widthName);
1484         std::cout << "名称";
1485         std::cout.width(5);
1486         std::cout << "类型";
1487         std::cout.width(5);
1488         std::cout << "学分";
1489         std::cout.width(5);

```



```

1490         std::cout << "人数";
1491         std::cout.width(5);
1492         std::cout << "均分";
1493         std::cout.width(5);
1494         std::cout << "均绩" << std::endl;
1495         for (cnt = 0; cnt < length; cnt++)
1496             head[cnt]->printInfo(widthName);
1497         delete[] head;
1498     }
1499     else
1500     {
1501         std::cout << "任选课信息为空！" << std::endl;
1502     }
1503     break;
1504     default:
1505         std::cout << "课程类型错误！" << std::endl;
1506         break;
1507     }
1508 }
1509
1510 // 学生排序函数，direction为1表示升序，为-1表示降序，keycol为关键列
1511 // 关键列：0-studentNo, 1-name, 2-totalScore, 3-totalCredit, 4-totalGPA, 5-
averageScore, 6-averageGPA
1512 void Database::sortStudentCustom(Student **head, int length, int direction, int
keycol)
1513 {
1514     Student *tmp;
1515     bool flag;
1516     do
1517     {
1518         flag = false;
1519         for (int i = 0; i < length - 1; i++)
1520         {
1521             if (compareStudent(head[i], head[i + 1], direction, keycol) < 0)
1522             {
1523                 tmp = head[i];
1524                 head[i] = head[i + 1];
1525                 head[i + 1] = tmp;
1526                 flag = true;
1527             }
1528         }
1529     } while (flag);
1530 }
1531
1532 // 课程排序函数，direction为1表示升序，为-1表示降序，keycol为关键列
1533 // 关键列：0-name, 1-credit, 2-studentNum, 3-averageScore, 4-averageGPA
1534 void Database::sortLectureCustom(Lecture **head, int length, int direction, int
keycol)
1535 {
1536     Lecture *tmp;
1537     bool flag;
1538     do

```

```

1539     {
1540         flag = false;
1541         for (int i = 0; i < length - 1; i++)
1542         {
1543             if (compareLecture(head[i], head[i + 1], direction, keycol) < 0)
1544             {
1545                 tmp = head[i];
1546                 head[i] = head[i + 1];
1547                 head[i + 1] = tmp;
1548                 flag = true;
1549             }
1550         }
1551     } while (flag);
1552 }
1553
1554 // 学生比较函数, direction为1表示升序, 为-1表示降序, keycol为关键列
1555 // 关键列: 0-studentNo, 1-name, 2-totalScore, 3-totalCredit, 4-totalGPA, 5-
1556 // averageScore, 6-averageGPA
1557 double Database::compareStudent(Student *a, Student *b, int direction, int keycol)
1558 {
1559     double compare = 0;
1560     switch (keycol)
1561     {
1562     case 0:
1563         compare = a->getStudentNo() - b->getStudentNo();
1564         break;
1565     case 1:
1566         compare = (a->getName() > b->getName());
1567         break;
1568     case 2:
1569         compare = a->getTotalScore() - b->getTotalScore();
1570         break;
1571     case 3:
1572         compare = a->getTotalCredit() - b->getTotalCredit();
1573         break;
1574     case 4:
1575         compare = a->getTotalGPA() - b->getTotalGPA();
1576         break;
1577     case 5:
1578         compare = a->getAverageScore() - b->getAverageScore();
1579         break;
1580     case 6:
1581         compare = a->getAverageGPA() - b->getAverageGPA();
1582         break;
1583     default:
1584         compare = 0;
1585         break;
1586     }
1587     return compare * direction;
1588 }
1589
1590 // 课程比较函数, direction为1表示升序, 为-1表示降序, keycol为关键列

```

```

1590 // 关键列: 0-name, 1-credit, 2-studentNum, 3-averageScore, 4-averageGPA
1591 double Database::compareLecture(Lecture *a, Lecture *b, int direction, int keycol)
1592 {
1593     double compare = 0;
1594     switch (keycol)
1595     {
1596     case 0:
1597         compare = (a->getName() > b->getName());
1598         break;
1599     case 1:
1600         compare = a->getCredit() - b->getCredit();
1601         break;
1602     case 2:
1603         compare = a->getStudentNum() - b->getStudentNum();
1604         break;
1605     case 3:
1606         compare = a->getAverageScore() - b->getAverageScore();
1607         break;
1608     case 4:
1609         compare = a->getAverageGPA() - b->getAverageGPA();
1610         break;
1611     default:
1612         compare = 0;
1613         break;
1614     }
1615     return compare * direction;
1616 }
1617
1618 // 打印全部学生信息
1619 void Database::printStudent()
1620 {
1621     std::ofstream out("output_student_all.md", std::ios::out | std::ios::trunc);
1622     if (!out.is_open())
1623     {
1624         std::cerr << "Error: cannot open file \""
1625             << "output_student_all.md"
1626             << "\"." << std::endl;
1627         return;
1628     }
1629     out << "<h1><center>xx大学学生成绩单</center></h1>" << std::endl
1630         << std::endl
1631         << "<h2><center>全部学生成绩</center></h2>" << std::endl
1632         << std::endl
1633         << "<table border='1' align='center'>" << std::endl
1634         << "<tr><th>学号</th><th>姓名</th><th>总学分</th><th>平均成绩</th><th>平均绩点"
1635         << std::endl;
1636         Student **head;
1637         int length = studentList.size();
1638         if (length > 0)
1639         {
1640             head = new Student *[length];
1641             Node<Student> *tmp = studentList.getFirst();

```

```

1641     int cnt = 0;
1642     do
1643     {
1644         head[cnt] = &tmp->data;
1645         tmp = studentList.getNext();
1646         cnt++;
1647     } while (tmp != nullptr);
1648     sortStudentCustom(head, length, -1, 0);
1649     for (cnt = 0; cnt < length; cnt++)
1650     {
1651         out << "<tr><td>" << head[cnt]->getStudentNo()
1652             << "</td><td>" << head[cnt]->getName()
1653             << "</td><td>" << head[cnt]->getTotalCredit()
1654             << "</td><td>" << head[cnt]->getAverageScore()
1655             << "</td><td>" << head[cnt]->getAverageGPA() << "</td></tr>" <<
std::endl;
1656     }
1657     delete[] head;
1658 }
1659 else
1660 {
1661     out << "<tr><td colspan=\\5\\>无学生信息</td></tr>" << std::endl;
1662 }
1663 out << "</table>" << std::endl
1664     << std::endl
1665     << "打印时间: " << timeStampToString(time(0)) << std::endl
1666     << std::endl
1667     << "****" << std::endl
1668     << std::endl
1669     << "<h1><center>成绩单说明</center></h1>" << std::endl
1670     << std::endl
1671     << "1. 本成绩单按照学号升序排列" << std::endl
1672     << "2. 本成绩单仅包含全部学生的总成绩信息" << std::endl
1673     << "3. 本成绩单仅供参考, 不作为最终成绩" << std::endl
1674     << "4. 本成绩单由学生成绩管理系统自动生成" << std::endl
1675     << "5. 本成绩单最终解释权归xx大学所有" << std::endl
1676     << std::endl
1677     << "****" << std::endl
1678     << std::endl
1679     << "<h1><center>成绩记载说明</center></h1>" << std::endl
1680     << std::endl
1681     << "<table border=\\1\\ align=\\center\\>" << std::endl
1682     << "<tr><th>等级制成绩</th><th>绩点</th><th>对应百分制成绩范围</th></tr>" <<
std::endl
1683     << "<tr><td>A+</td><td rowspan=\\3\\>4.0</td><td rowspan=\\2\\>95~100</td>
</tr>" << std::endl
1684     << "<tr><td>A</td></tr>" << std::endl
1685     << "<tr><td>A-</td><td>90~94</td></tr>" << std::endl
1686     << "<tr><td>B+</td><td>3.6</td><td>85~89</td></tr>" << std::endl
1687     << "<tr><td>B</td><td>3.3</td><td>80~84</td></tr>" << std::endl
1688     << "<tr><td>B-</td><td>3.0</td><td>77~79</td></tr>" << std::endl
1689     << "<tr><td>C+</td><td>2.6</td><td>73~76</td></tr>" << std::endl

```

```

1690     << "<tr><td>C</td><td>2.3</td><td>70~72</td></tr>" << std::endl
1691     << "<tr><td>C-</td><td>2.0</td><td>67~69</td></tr>" << std::endl
1692     << "<tr><td>D+</td><td>1.6</td><td>63~66</td></tr>" << std::endl
1693     << "<tr><td>D</td><td>1.3</td><td>60~62</td></tr>" << std::endl
1694     << "<tr><td>F</td><td>0</td><td>0~59</td></tr>" << std::endl
1695     << "</table>" << std::endl
1696     << std::endl
1697     << "平均学分绩（GPA）的计算方法为:  $GPA = \frac{\sum \text{课程学分} * \text{绩点}}{\sum \text{课程学分}}$ " << std::endl
1698     << std::endl;
1699     out.close();
1700     std::cout << "打印成功! " << std::endl;
1701 }
1702
1703 // 按学号打印学生信息
1704 void Database::printStudent(int studentNo)
1705 {
1706     Student *stu = findStudent(studentNo);
1707     if (stu == nullptr)
1708     {
1709         std::cout << "未找到该学生! " << std::endl;
1710         return;
1711     }
1712     std::string filename = "output_student_" + std::to_string(studentNo) + "_" + stu-
>getName() + ".md";
1713     std::ofstream out(filename, std::ios::out | std::ios::trunc);
1714     if (!out.is_open())
1715     {
1716         std::cerr << "Error: cannot open file \"" << filename << "\"." << std::endl;
1717         return;
1718     }
1719     out << "<h1><center>xx大学学生成绩单</center></h1>" << std::endl
1720     << std::endl
1721     << "学号: " << studentNo << std::endl
1722     << std::endl
1723     << "姓名: " << stu->getName() << std::endl
1724     << std::endl
1725     << "总学分: " << stu->getTotalCredit() << std::endl
1726     << std::endl
1727     << "平均成绩: " << stu->getAverageScore() << std::endl
1728     << std::endl
1729     << "平均绩点: " << stu->getAverageGPA() << std::endl
1730     << std::endl
1731     << "<table border='1' align='center'>" << std::endl
1732     << "<tr><th>课程名称</th><th>课程类型</th><th>学分</th><th>成绩</th><th>绩点</th></tr>" << std::endl;
1733     if (stu->getLectureNum() > 0)
1734     {
1735         for (int i = 0; i < stu->getLectureNum(); i++)
1736         {
1737             if (!stu->getLecturePF()[i])
1738             {

```

```

1739         out << "<tr><td>" << stu->getLectureName()[i]
1740             << "</td><td>" << ((stu->getLectureType()[i] == REQUIRED) ? "必选
课" : ((stu->getLectureType()[i] == LIMITED) ? "限选课" : "任选课"))
1741             << "</td><td>" << stu->getLectureCredit()[i]
1742             << "</td><td>" << stu->getLectureScore()[i]
1743             << "</td><td>" << stu->getLectureGPA()[i] << "</td></tr>" <<
std::endl;
1744     }
1745     else
1746     {
1747         out << "<tr><td>" << stu->getLectureName()[i]
1748             << "</td><td>" << ((stu->getLectureType()[i] == REQUIRED) ? "必选
课" : ((stu->getLectureType()[i] == LIMITED) ? "限选课" : "任选课"))
1749             << "</td><td>" << stu->getLectureCredit()[i]
1750             << "</td><td>"
1751             << "N/A"
1752             << "</td><td>" << (stu->getLectureGPA()[i] ? "P" : "F") << "</td>
</tr>" << std::endl;
1753     }
1754 }
1755 }
1756 else
1757 {
1758     out << "<tr><td colspan=\\"5\\">无</td></tr>" << std::endl;
1759 }
1760 out << "</table>" << std::endl
1761     << std::endl
1762     << "打印时间: " << timeStampToString(time(0)) << std::endl
1763     << std::endl
1764     << "****" << std::endl
1765     << std::endl
1766     << "<h1><center>成绩单说明</center></h1>" << std::endl
1767     << std::endl
1768     << "1. 本成绩单仅包含该学生的各课程成绩信息" << std::endl
1769     << "2. 本成绩单仅供参考, 不作为最终成绩" << std::endl
1770     << "3. 本成绩单由学生成绩管理系统自动生成" << std::endl
1771     << "4. 本成绩单最终解释权归××大学所有" << std::endl
1772     << std::endl
1773     << "****" << std::endl
1774     << std::endl
1775     << "<h1><center>成绩记载说明</center></h1>" << std::endl
1776     << std::endl
1777     << "<table border=\\"1\\" align=\\"center\\">" << std::endl
1778     << "<tr><th>等级制成绩</th><th>绩点</th><th>对应百分制成绩范围</th></tr>" <<
std::endl
1779     << "<tr><td>A+</td><td rowspan=\\"3\\">4.0</td><td rowspan=\\"2\\">95~100</td>
</tr>" << std::endl
1780     << "<tr><td>A</td></tr>" << std::endl
1781     << "<tr><td>A-</td><td>90~94</td></tr>" << std::endl
1782     << "<tr><td>B+</td><td>3.6</td><td>85~89</td></tr>" << std::endl
1783     << "<tr><td>B</td><td>3.3</td><td>80~84</td></tr>" << std::endl
1784     << "<tr><td>B-</td><td>3.0</td><td>77~79</td></tr>" << std::endl

```

```

1785     << "<tr><td>C+</td><td>2.6</td><td>73~76</td></tr>" << std::endl
1786     << "<tr><td>C</td><td>2.3</td><td>70~72</td></tr>" << std::endl
1787     << "<tr><td>C-</td><td>2.0</td><td>67~69</td></tr>" << std::endl
1788     << "<tr><td>D+</td><td>1.6</td><td>63~66</td></tr>" << std::endl
1789     << "<tr><td>D</td><td>1.3</td><td>60~62</td></tr>" << std::endl
1790     << "<tr><td>F</td><td>0</td><td>0~59</td></tr>" << std::endl
1791     << "</table>" << std::endl
1792     << std::endl
1793     << "平均学分绩（GPA）的计算方法为：$GPA=\frac{\Sigma \text{课程学分} * \text{绩点}}{\Sigma \text{课程学分}}$" << std::endl
1794     << std::endl;
1795     out.close();
1796     std::cout << "打印成功！" << std::endl;
1797 }
1798
1799 // 按姓名打印学生信息
1800 void Database::printStudent(const std::string &name)
1801 {
1802     Student *stu = findStudent(name);
1803     if (stu == nullptr)
1804     {
1805         std::cout << "未找到该学生！" << std::endl;
1806         return;
1807     }
1808     std::string filename = "output_student_" + std::to_string(stu->getStudentNo()) +
1809     "_" + name + ".md";
1809     std::ofstream out(filename, std::ios::out | std::ios::trunc);
1810     if (!out.is_open())
1811     {
1812         std::cerr << "Error: cannot open file \"" << filename << "\"." << std::endl;
1813         return;
1814     }
1815     out << "<h1><center>xx大学学生成绩单</center></h1>" << std::endl
1816     << std::endl
1817     << "学号：" << stu->getStudentNo() << std::endl
1818     << std::endl
1819     << "姓名：" << name << std::endl
1820     << std::endl
1821     << "总学分：" << stu->getTotalCredit() << std::endl
1822     << std::endl
1823     << "平均成绩：" << stu->getAverageScore() << std::endl
1824     << std::endl
1825     << "平均绩点：" << stu->getAverageGPA() << std::endl
1826     << std::endl
1827     << "<table border='1' align='center'>" << std::endl
1828     << "<tr><th>课程名称</th><th>课程类型</th><th>学分</th><th>成绩</th><th>绩点</th></tr>" << std::endl;
1829     if (stu->getLectureNum() > 0)
1830     {
1831         for (int i = 0; i < stu->getLectureNum(); i++)
1832         {
1833             if (!stu->getLecturePF()[i])

```

```

1834         {
1835             out << "<tr><td>" << stu->getLectureName()[i]
1836                 << "</td><td>" << ((stu->getLectureType()[i] == REQUIRED) ? "必修课" : ((stu->getLectureType()[i] == LIMITED) ? "限选课" : "任选课"))
1837                 << "</td><td>" << stu->getLectureCredit()[i]
1838                 << "</td><td>" << stu->getLectureScore()[i]
1839                 << "</td><td>" << stu->getLectureGPA()[i] << "</td></tr>" <<
std::endl;
1840         }
1841         else
1842         {
1843             out << "<tr><td>" << stu->getLectureName()[i]
1844                 << "</td><td>" << ((stu->getLectureType()[i] == REQUIRED) ? "必修课" : ((stu->getLectureType()[i] == LIMITED) ? "限选课" : "任选课"))
1845                 << "</td><td>" << stu->getLectureCredit()[i]
1846                 << "</td><td>"
1847                 << "N/A"
1848                 << "</td><td>" << (stu->getLectureGPA()[i] ? "P" : "F") << "</td>
</tr>" << std::endl;
1849         }
1850     }
1851 }
1852 else
1853 {
1854     out << "<tr><td colspan=\\5\\>无</td></tr>" << std::endl;
1855 }
1856 out << "</table>" << std::endl
1857     << std::endl
1858     << "打印时间: " << timeStampToString(time(0)) << std::endl
1859     << std::endl
1860     << "****" << std::endl
1861     << std::endl
1862     << "<h1><center>成绩单说明</center></h1>" << std::endl
1863     << std::endl
1864     << "1. 本成绩单仅包含该学生的各课程成绩信息" << std::endl
1865     << "2. 本成绩单仅供参考, 不作为最终成绩" << std::endl
1866     << "3. 本成绩单由学生成绩管理系统自动生成" << std::endl
1867     << "4. 本成绩单最终解释权归xx大学所有" << std::endl
1868     << std::endl
1869     << "****" << std::endl
1870     << std::endl
1871     << "<h1><center>成绩记载说明</center></h1>" << std::endl
1872     << std::endl
1873     << "<table border=\\1\\ align=\\center\\>" << std::endl
1874     << "<tr><th>等级制成绩</th><th>绩点</th><th>对应百分制成绩范围</th></tr>" <<
std::endl
1875     << "<tr><td>A+</td><td rowspan=\\3\\>4.0</td><td rowspan=\\2\\>95~100</td>
</tr>" << std::endl
1876     << "<tr><td>A</td></tr>" << std::endl
1877     << "<tr><td>A-</td><td>90~94</td></tr>" << std::endl
1878     << "<tr><td>B+</td><td>3.6</td><td>85~89</td></tr>" << std::endl
1879     << "<tr><td>B</td><td>3.3</td><td>80~84</td></tr>" << std::endl

```



```

1880     << "<tr><td>B-</td><td>3.0</td><td>77~79</td></tr>" << std::endl
1881     << "<tr><td>C+</td><td>2.6</td><td>73~76</td></tr>" << std::endl
1882     << "<tr><td>C</td><td>2.3</td><td>70~72</td></tr>" << std::endl
1883     << "<tr><td>C-</td><td>2.0</td><td>67~69</td></tr>" << std::endl
1884     << "<tr><td>D+</td><td>1.6</td><td>63~66</td></tr>" << std::endl
1885     << "<tr><td>D</td><td>1.3</td><td>60~62</td></tr>" << std::endl
1886     << "<tr><td>F</td><td>0</td><td>0~59</td></tr>" << std::endl
1887     << "</table>" << std::endl
1888     << std::endl
1889     << "平均学分绩（GPA）的计算方法为：$GPA=\frac{\Sigma \text{课程学分} * \text{绩点}}{\Sigma \text{课程学分}}$" << std::endl
1890     << std::endl;
1891     out.close();
1892     std::cout << "打印成功！" << std::endl;
1893 }
1894
1895 // 打印全部课程信息
1896 void Database::printLecture()
1897 {
1898     std::ofstream out("output_lecture_all.md", std::ios::out | std::ios::trunc);
1899     if (!out.is_open())
1900     {
1901         std::cerr << "Error: cannot open file \""
1902                 << "output_lecture_all.md"
1903                 << "\"." << std::endl;
1904         return;
1905     }
1906
1907     out << "<h1><center>xx大学学生成绩单</center></h1>" << std::endl
1908         << std::endl
1909         << "<h2><center>全部课程信息</center></h2>" << std::endl
1910         << std::endl
1911         << "<table border=\"1\" align=\"center\">" << std::endl
1912         << "<tr><th>名称</th><th>类型</th><th>学分</th><th>学生数</th><th>平均成绩</th><th>平均绩点</th></tr>" << std::endl;
1913     Lecture **head;
1914     int reqLength = requiredList.size(), limLength = limitedList.size(), optLength = optionalList.size();
1915     if (reqLength + limLength + optLength > 0)
1916     {
1917         head = new Lecture *[reqLength + limLength + optLength];
1918         int cnt = 0;
1919         if (reqLength > 0)
1920         {
1921             Node<LectureRequired> *tmp = requiredList.getFirst();
1922             do
1923             {
1924                 head[cnt] = &tmp->data;
1925                 tmp = requiredList.getNext();
1926                 cnt++;
1927             } while (tmp != nullptr);
1928         }

```

```

1929     if (limLength > 0)
1930     {
1931         Node<LectureLimited> *tmp = limitedList.getFirst();
1932         do
1933         {
1934             head[cnt] = &tmp->data;
1935             tmp = limitedList.getNext();
1936             cnt++;
1937         } while (tmp != nullptr);
1938     }
1939     if (optLength > 0)
1940     {
1941         Node<LectureOptional> *tmp = optionalList.getFirst();
1942         do
1943         {
1944             head[cnt] = &tmp->data;
1945             tmp = optionalList.getNext();
1946             cnt++;
1947         } while (tmp != nullptr);
1948     }
1949     sortLectureCustom(head, reqLength + limLength + optLength, -1, 0);
1950     for (cnt = 0; cnt < reqLength + limLength + optLength; cnt++)
1951     {
1952         out << "<tr><td>" << head[cnt]->getName()
1953             << "</td><td>" << ((head[cnt]->getLectureType() == REQUIRED) ? "必修课" : (head[cnt]->getLectureType() == LIMITED ? "限选课" : "任选课"))
1954             << "</td><td>" << head[cnt]->getCredit()
1955             << "</td><td>" << head[cnt]->getStudentNum()
1956             << "</td><td>" << head[cnt]->getAverageScore()
1957             << "</td><td>" << head[cnt]->getAverageGPA() << "</td></tr>" <<
std::endl;
1958     }
1959     delete[] head;
1960 }
1961 else
1962 {
1963     out << "<tr><td colspan=\\"6\\>无课程信息</td></tr>" << std::endl;
1964 }
1965 out << "</table>" << std::endl
1966     << std::endl
1967     << "打印时间: " << timeStampToString(time(0)) << std::endl
1968     << std::endl
1969     << "****" << std::endl
1970     << std::endl
1971     << "<h1><center>成绩单说明</center></h1>" << std::endl
1972     << std::endl
1973     << "1. 本成绩单按照课程名称升序排列" << std::endl
1974     << "2. 本成绩单仅包含全部课程的总成绩信息" << std::endl
1975     << "3. 本成绩单仅供参考, 不作为最终成绩" << std::endl
1976     << "4. 本成绩单由学生成绩管理系统自动生成" << std::endl
1977     << "5. 本成绩单最终解释权归xx大学所有" << std::endl
1978     << std::endl

```

```

1979     << "****" << std::endl
1980     << std::endl
1981     << "<h1><center>成绩记载说明</center></h1>" << std::endl
1982     << std::endl
1983     << "<table border=\"1\" align=\"center\">" << std::endl
1984     << "<tr><th>等级制成绩</th><th>绩点</th><th>对应百分制成绩范围</th></tr>" <<
std::endl
1985     << "<tr><td>A+</td><td rowspan=\"3\">4.0</td><td rowspan=\"2\">95~100</td>
</tr>" << std::endl
1986     << "<tr><td>A</td></tr>" << std::endl
1987     << "<tr><td>A-</td><td>90~94</td></tr>" << std::endl
1988     << "<tr><td>B+</td><td>3.6</td><td>85~89</td></tr>" << std::endl
1989     << "<tr><td>B</td><td>3.3</td><td>80~84</td></tr>" << std::endl
1990     << "<tr><td>B-</td><td>3.0</td><td>77~79</td></tr>" << std::endl
1991     << "<tr><td>C+</td><td>2.6</td><td>73~76</td></tr>" << std::endl
1992     << "<tr><td>C</td><td>2.3</td><td>70~72</td></tr>" << std::endl
1993     << "<tr><td>C-</td><td>2.0</td><td>67~69</td></tr>" << std::endl
1994     << "<tr><td>D+</td><td>1.6</td><td>63~66</td></tr>" << std::endl
1995     << "<tr><td>D</td><td>1.3</td><td>60~62</td></tr>" << std::endl
1996     << "<tr><td>F</td><td>0</td><td>0~59</td></tr>" << std::endl
1997     << "</table>" << std::endl
1998     << std::endl
1999     << "平均学分绩（GPA）的计算方法为：$GPA=\frac{\Sigma \text{课程学分} * \text{绩点}}{\Sigma \text{课程学分}}$" << std::endl
2000     << std::endl;
2001     out.close();
2002     std::cout << "打印成功！" << std::endl;
2003 }
2004
2005 // 按类型打印课程信息
2006 void Database::printLecture(LectureType type)
2007 {
2008     std::string filename = "output_lecture_" + std::string((type == REQUIRED) ?
"required" : (type == LIMITED ? "limited" : "optional")) + ".md";
2009     std::ofstream out(filename, std::ios::out | std::ios::trunc);
2010     if (!out.is_open())
2011     {
2012         std::cerr << "Error: cannot open file \"" << filename << "\"." << std::endl;
2013         return;
2014     }
2015     Lecture **head = nullptr;
2016     int length = 0;
2017     int cnt = 0;
2018     switch (type)
2019     {
2020     case REQUIRED:
2021         out << "<h1><center>××大学学生成绩单</center></h1>" << std::endl
2022         << std::endl
2023         << "<h2><center>必选课程信息</center></h2>" << std::endl
2024         << std::endl
2025         << "<table border=\"1\" align=\"center\">" << std::endl

```

```

2026         << "<tr><th>名称</th><th>类型</th><th>学分</th><th>学生数</th><th>平均成绩
2027         </th><th>平均绩点</th></tr>" << std::endl;
2028         length = requiredList.size();
2029         head = new Lecture *[length];
2030         if (length > 0)
2031         {
2032             Node<LectureRequired> *tmp = requiredList.getFirst();
2033             do
2034             {
2035                 head[cnt] = &tmp->data;
2036                 tmp = requiredList.getNext();
2037                 cnt++;
2038             } while (tmp != nullptr);
2039         }
2040         break;
2041     case LIMITED:
2042         out << "<h1><center>xx大学学生成绩单</center></h1>" << std::endl
2043         << std::endl
2044         << "<h2><center>限选课程信息</center></h2>" << std::endl
2045         << std::endl
2046         << "<table border=\"1\" align=\"center\">" << std::endl
2047         << "<tr><th>名称</th><th>类型</th><th>学分</th><th>学生数</th><th>平均成绩
2048         </th><th>平均绩点</th></tr>" << std::endl;
2049         length = limitedList.size();
2050         head = new Lecture *[length];
2051         if (length > 0)
2052         {
2053             Node<LectureLimited> *tmp = limitedList.getFirst();
2054             do
2055             {
2056                 head[cnt] = &tmp->data;
2057                 tmp = limitedList.getNext();
2058                 cnt++;
2059             } while (tmp != nullptr);
2060         }
2061         break;
2062     case OPTIONAL:
2063         out << "<h1><center>xx大学学生成绩单</center></h1>" << std::endl
2064         << std::endl
2065         << "<h2><center>任选课程信息</center></h2>" << std::endl
2066         << std::endl
2067         << "<table border=\"1\" align=\"center\">" << std::endl
2068         << "<tr><th>名称</th><th>类型</th><th>学分</th><th>学生数</th><th>平均成绩
2069         </th><th>平均绩点</th></tr>" << std::endl;
2070         length = optionalList.size();
2071         head = new Lecture *[length];
2072         if (length > 0)
2073         {
2074             Node<LectureOptional> *tmp = optionalList.getFirst();
2075             do
2076             {
2077                 head[cnt] = &tmp->data;

```

```

2075         tmp = optionalList.getNext();
2076         cnt++;
2077     } while (tmp != nullptr);
2078 }
2079 break;
2080 default:
2081     return;
2082     break;
2083 }
2084 if (length > 0)
2085 {
2086     sortLectureCustom(head, length, -1, 0);
2087     for (cnt = 0; cnt < length; cnt++)
2088     {
2089         out << "<tr><td>" << head[cnt]->getName()
2090             << "</td><td>" << ((head[cnt]->getLectureType() == REQUIRED) ? "必选
课" : (head[cnt]->getLectureType() == LIMITED ? "限选课" : "任选课"))
2091             << "</td><td>" << head[cnt]->getCredit()
2092             << "</td><td>" << head[cnt]->getStudentNum()
2093             << "</td><td>" << head[cnt]->getAverageScore()
2094             << "</td><td>" << head[cnt]->getAverageGPA() << "</td></tr>" <<
std::endl;
2095     }
2096     delete[] head;
2097 }
2098 else
2099 {
2100     out << "<tr><td colspan=\\\"6\\\">无课程信息</td></tr>" << std::endl;
2101 }
2102 out << "</table>" << std::endl
2103     << std::endl
2104     << "打印时间: " << timeStampToString(time(0)) << std::endl
2105     << std::endl
2106     << "****" << std::endl
2107     << std::endl
2108     << "<h1><center>成绩单说明</center></h1>" << std::endl
2109     << std::endl
2110     << "1. 本成绩单按照课程名称升序排列" << std::endl
2111     << "2. 本成绩单仅包含全部课程的总成绩信息" << std::endl
2112     << "3. 本成绩单仅供参考，不作为最终成绩" << std::endl
2113     << "4. 本成绩单由学生成绩管理系统自动生成" << std::endl
2114     << "5. 本成绩单最终解释权归xx大学所有" << std::endl
2115     << std::endl
2116     << "****" << std::endl
2117     << std::endl
2118     << "<h1><center>成绩记载说明</center></h1>" << std::endl
2119     << std::endl
2120     << "<table border=\\\"1\\\" align=\\\"center\\\">" << std::endl
2121     << "<tr><th>等级制成绩</th><th>绩点</th><th>对应百分制成绩范围</th></tr>" <<
std::endl
2122     << "<tr><td>A+</td><td rowspan=\\\"3\\\">4.0</td><td rowspan=\\\"2\\\">95~100</td>
</tr>" << std::endl

```

```

2123         << "<tr><td>A</td></tr>" << std::endl
2124         << "<tr><td>A-</td><td>90~94</td></tr>" << std::endl
2125         << "<tr><td>B+</td><td>3.6</td><td>85~89</td></tr>" << std::endl
2126         << "<tr><td>B</td><td>3.3</td><td>80~84</td></tr>" << std::endl
2127         << "<tr><td>B-</td><td>3.0</td><td>77~79</td></tr>" << std::endl
2128         << "<tr><td>C+</td><td>2.6</td><td>73~76</td></tr>" << std::endl
2129         << "<tr><td>C</td><td>2.3</td><td>70~72</td></tr>" << std::endl
2130         << "<tr><td>C-</td><td>2.0</td><td>67~69</td></tr>" << std::endl
2131         << "<tr><td>D+</td><td>1.6</td><td>63~66</td></tr>" << std::endl
2132         << "<tr><td>D</td><td>1.3</td><td>60~62</td></tr>" << std::endl
2133         << "<tr><td>F</td><td>0</td><td>0~59</td></tr>" << std::endl
2134         << "</table>" << std::endl
2135         << std::endl
2136         << "平均学分绩（GPA）的计算方法为：$GPA=\frac{\Sigma \text{课程学分} * \text{绩点}}{\Sigma \text{课程学分}}$" << std::endl
2137         << std::endl;
2138     out.close();
2139     std::cout << "打印成功！" << std::endl;
2140 }
2141
2142 // 按课程名打印课程信息
2143 void Database::printLecture(const std::string &name)
2144 {
2145     Lecture *lec = findLecture(name);
2146     if (lec == nullptr)
2147     {
2148         std::cout << "未找到该课程！" << std::endl;
2149         return;
2150     }
2151     std::string filename = "output_lecture_" + name + ".md";
2152     std::ofstream out(filename, std::ios::out | std::ios::trunc);
2153     if (!out.is_open())
2154     {
2155         std::cerr << "Error: cannot open file \"" << filename << "\"." << std::endl;
2156         return;
2157     }
2158     out << "<h1><center>xx大学学生成绩单</center></h1>" << std::endl
2159         << std::endl
2160         << "名称: " << name << std::endl
2161         << std::endl
2162         << "学分: " << lec->getCredit() << std::endl
2163         << std::endl
2164         << "平均成绩: " << lec->getAverageScore() << std::endl
2165         << std::endl
2166         << "平均绩点: " << lec->getAverageGPA() << std::endl
2167         << std::endl
2168         << "<table border='1' align='center'>" << std::endl
2169         << "<tr><th>学号</th><th>姓名</th><th>成绩</th><th>绩点</th></tr>" <<
std::endl;
2170     if (lec->getStudentNum() > 0)
2171     {
2172         for (int i = 0; i < lec->getStudentNum(); i++)

```

```

2173     {
2174         if (!queryOptional(name, false))
2175         {
2176             out << "<tr><td>" << lec->getStudentNo()[i]
2177                 << "</td><td>" << lec->getStudentName()[i]
2178                 << "</td><td>" << lec->getStudentScore()[i]
2179                 << "</td><td>" << lec->getStudentGPA()[i] << "</td></tr>" <<
std::endl;
2180         }
2181         else if (!findOptional(name)->getStudentPF()[i])
2182         {
2183             out << "<tr><td>" << lec->getStudentNo()[i]
2184                 << "</td><td>" << lec->getStudentName()[i]
2185                 << "</td><td>" << lec->getStudentScore()[i]
2186                 << "</td><td>" << lec->getStudentGPA()[i] << "</td></tr>" <<
std::endl;
2187         }
2188         else
2189         {
2190             out << "<tr><td>" << lec->getStudentNo()[i]
2191                 << "</td><td>" << lec->getStudentName()[i]
2192                 << "</td><td>"
2193                 << "N/A"
2194                 << "</td><td>" << (lec->getStudentGPA()[i] ? "P" : "F") << "</td>
</tr>" << std::endl;
2195         }
2196     }
2197 }
2198 else
2199 {
2200     out << "<tr><td colspan=\\"4\\">无学生选修该课程</td></tr>" << std::endl;
2201 }
2202 out << "</table>" << std::endl
2203     << std::endl
2204     << "打印时间: " << timeStampToString(time(0)) << std::endl
2205     << std::endl
2206     << "****" << std::endl
2207     << std::endl
2208     << "<h1><center>成绩单说明</center></h1>" << std::endl
2209     << std::endl
2210     << "1. 本成绩单仅包含该课程的各学生成绩信息" << std::endl
2211     << "2. 本成绩单仅供参考, 不作为最终成绩" << std::endl
2212     << "3. 本成绩单由学生成绩管理系统自动生成" << std::endl
2213     << "4. 本成绩单最终解释权归xx大学所有" << std::endl
2214     << std::endl
2215     << "****" << std::endl
2216     << std::endl
2217     << "<h1><center>成绩记载说明</center></h1>" << std::endl
2218     << std::endl
2219     << "<table border=\\"1\\" align=\\"center\\">" << std::endl
2220     << "<tr><th>等级制成绩</th><th>绩点</th><th>对应百分制成绩范围</th></tr>" <<
std::endl

```

```

2221         << "<tr><td>A+</td><td rowspan=\"3\">4.0</td><td rowspan=\"2\">95~100</td>
</tr>" << std::endl
2222         << "<tr><td>A-</td></tr>" << std::endl
2223         << "<tr><td>A-</td><td>90~94</td></tr>" << std::endl
2224         << "<tr><td>B+</td><td>3.6</td><td>85~89</td></tr>" << std::endl
2225         << "<tr><td>B-</td><td>3.3</td><td>80~84</td></tr>" << std::endl
2226         << "<tr><td>B-</td><td>3.0</td><td>77~79</td></tr>" << std::endl
2227         << "<tr><td>C+</td><td>2.6</td><td>73~76</td></tr>" << std::endl
2228         << "<tr><td>C-</td><td>2.3</td><td>70~72</td></tr>" << std::endl
2229         << "<tr><td>C-</td><td>2.0</td><td>67~69</td></tr>" << std::endl
2230         << "<tr><td>D+</td><td>1.6</td><td>63~66</td></tr>" << std::endl
2231         << "<tr><td>D-</td><td>1.3</td><td>60~62</td></tr>" << std::endl
2232         << "<tr><td>F</td><td>0</td><td>0~59</td></tr>" << std::endl
2233         << "</table>" << std::endl
2234         << std::endl
2235         << "平均学分绩（GPA）的计算方法为：$GPA=\frac{\Sigma \text{课程学分} * \text{绩点}}{\Sigma \text{课程学分}}$" << std::endl
2236         << std::endl;
2237         out.close();
2238         std::cout << "打印成功！" << std::endl;
2239     }
2240
2241     // 百分制成绩转化为绩点
2242     double Database::calculateGPA(int score)
2243     {
2244         double ret = 0;
2245         if (score >= 90)
2246             ret = 4.0;
2247         else if (score >= 85)
2248             ret = 3.6;
2249         else if (score >= 80)
2250             ret = 3.3;
2251         else if (score >= 77)
2252             ret = 3.0;
2253         else if (score >= 73)
2254             ret = 2.6;
2255         else if (score >= 70)
2256             ret = 2.3;
2257         else if (score >= 67)
2258             ret = 2.0;
2259         else if (score >= 63)
2260             ret = 1.6;
2261         else if (score >= 60)
2262             ret = 1.3;
2263         else
2264             ret = 0;
2265         return ret;
2266     }
2267
2268     // 从课程更新学生信息
2269     void Database::updateStudent()
2270     {

```



```

2271     studentList.makeEmpty();
2272     Node<LectureRequired> *req = requiredList.getFirst();
2273     Node<LectureLimited> *lim = limitedList.getFirst();
2274     Node<LectureOptional> *opt = optionalList.getFirst();
2275     if (req != nullptr)
2276     {
2277         do
2278         {
2279             for (int i = 0; i < req->data.getStudentNum(); i++)
2280             {
2281                 if (!queryStudent(req->data.getStudentNo()[i], false))
2282                 {
2283                     Node<Student> *stu = new Node<Student>;
2284                     stu->data.setStudentNo(req->data.getStudentNo()[i]);
2285                     stu->data.setName(req->data.getStudentName()[i]);
2286                     stu->data.setLectureNum(stu->data.getLectureNum() + 1);
2287                     stu->data.getLectureName().push_back(req->data.getName());
2288                     stu->data.getLectureType().push_back(REQUIRED);
2289                     stu->data.getLectureCredit().push_back(req->data.getCredit());
2290                     stu->data.getLectureScore().push_back(req->data.getStudentScore()
2291 [i]);
2292                     stu->data.getLectureGPA().push_back(req->data.getStudentGPA()
2293 [i]);
2294                     stu->data.getLecturePF().push_back(false);
2295                     stu->data.setTotalScore(stu->data.getTotalScore() + req-
2296 >data.getStudentScore()[i] * req->data.getCredit());
2297                     stu->data.setTotalCredit(stu->data.getTotalCredit() + req-
2298 >data.getCredit());
2299                     stu->data.setTotalGPA(stu->data.getTotalGPA() + calculateGPA(req-
2300 >data.getStudentScore()[i]) * req->data.getCredit());
2301                     if (stu->data.getTotalCredit() - stu->data.getTotalPFCredit())
2302                         stu->data.setAverageScore(stu->data.getTotalScore() / (stu-
2303 >data.getTotalCredit() - stu->data.getTotalPFCredit()));
2304                     if (stu->data.getTotalCredit() - stu->data.getTotalPFCredit())
2305                         stu->data.setAverageGPA(stu->data.getTotalGPA() / (stu-
2306 >data.getTotalCredit() - stu->data.getTotalPFCredit()));
2307                     studentList.insert_end(stu);
2308                 }
2309             }
2310         }
2311         else
2312         {
2313             Student *stu = findStudent(req->data.getStudentNo()[i]);
2314             stu->setLectureNum(stu->getLectureNum() + 1);
2315             stu->getLectureName().push_back(req->data.getName());
2316             stu->getLectureType().push_back(REQUIRED);
2317             stu->getLectureCredit().push_back(req->data.getCredit());
2318             stu->getLectureScore().push_back(req->data.getStudentScore()[i]);
2319             stu->getLectureGPA().push_back(req->data.getStudentGPA()[i]);
2320             stu->getLecturePF().push_back(false);
2321             stu->setTotalScore(stu->getTotalScore() + req-
2322 >data.getStudentScore()[i] * req->data.getCredit());
2323             stu->setTotalCredit(stu->getTotalCredit() + req-
2324 >data.getCredit());

```

```

2314         stu->setTotalGPA(stu->getTotalGPA() + calculateGPA(req-
>data.getStudentScore()[i]) * req->data.getCredit());
2315         if (stu->getTotalCredit() - stu->getTotalPFCredit())
2316             stu->setAverageScore(stu->getTotalScore() / (stu-
>getTotalCredit() - stu->getTotalPFCredit()));
2317         if (stu->getTotalCredit() - stu->getTotalPFCredit())
2318             stu->setAverageGPA(stu->getTotalGPA() / (stu-
>getTotalCredit() - stu->getTotalPFCredit()));
2319     }
2320 }
2321 req = requiredList.getNext();
2322 } while (req != nullptr);
2323 }
2324 if (lim != nullptr)
2325 {
2326     do
2327     {
2328         for (int i = 0; i < lim->data.getStudentNum(); i++)
2329         {
2330             if (!queryStudent(lim->data.getStudentNo()[i], false))
2331             {
2332                 Node<Student> *stu = new Node<Student>;
2333                 stu->data.setStudentNo(lim->data.getStudentNo()[i]);
2334                 stu->data.setName(lim->data.getStudentName()[i]);
2335                 stu->data.setLectureNum(stu->data.getLectureNum() + 1);
2336                 stu->data.getLectureName().push_back(lim->data.getName());
2337                 stu->data.getLectureType().push_back(LIMITED);
2338                 stu->data.getLectureCredit().push_back(lim->data.getCredit());
2339                 stu->data.getLectureScore().push_back(lim->data.getStudentScore()
[i]);
2340                 stu->data.getLectureGPA().push_back(lim->data.getStudentGPA()
[i]);
2341                 stu->data.getLecturePF().push_back(false);
2342                 stu->data.setTotalScore(stu->data.getTotalScore() + lim-
>data.getStudentScore()[i] * lim->data.getCredit());
2343                 stu->data.setTotalCredit(stu->data.getTotalCredit() + lim-
>data.getCredit());
2344                 stu->data.setTotalGPA(stu->data.getTotalGPA() + calculateGPA(lim-
>data.getStudentScore()[i]) * lim->data.getCredit());
2345                 if (stu->data.getTotalCredit() - stu->data.getTotalPFCredit())
2346                     stu->data.setAverageScore(stu->data.getTotalScore() / (stu-
>data.getTotalCredit() - stu->data.getTotalPFCredit()));
2347                 if (stu->data.getTotalCredit() - stu->data.getTotalPFCredit())
2348                     stu->data.setAverageGPA(stu->data.getTotalGPA() / (stu-
>data.getTotalCredit() - stu->data.getTotalPFCredit()));
2349                 studentList.insert_end(stu);
2350             }
2351         } else
2352         {
2353             Student *stu = findStudent(lim->data.getStudentNo()[i]);
2354             stu->setLectureNum(stu->getLectureNum() + 1);
2355             stu->getLectureName().push_back(lim->data.getName());

```

```

2356         stu->getLectureType().push_back(LIMITED);
2357         stu->getLectureCredit().push_back(lim->data.getCredit());
2358         stu->getLectureScore().push_back(lim->data.getStudentScore()[i]);
2359         stu->getLectureGPA().push_back(lim->data.getStudentGPA()[i]);
2360         stu->getLecturePF().push_back(false);
2361         stu->setTotalScore(stu->getTotalScore() + lim-
>data.getStudentScore()[i] * lim->data.getCredit());
2362         stu->setTotalCredit(stu->getTotalCredit() + lim-
>data.getCredit());
2363         stu->setTotalGPA(stu->getTotalGPA() + calculateGPA(lim-
>data.getStudentScore()[i]) * lim->data.getCredit());
2364         if (stu->getTotalCredit() - stu->getTotalPFCredit())
2365             stu->setAverageScore(stu->getTotalScore() / (stu-
>getTotalCredit() - stu->getTotalPFCredit()));
2366         if (stu->getTotalCredit() - stu->getTotalPFCredit())
2367             stu->setAverageGPA(stu->getTotalGPA() / (stu-
>getTotalCredit() - stu->getTotalPFCredit()));
2368     }
2369 }
2370     lim = limitedList.getNext();
2371 } while (lim != nullptr);
2372 }
2373 if (opt != nullptr)
2374 {
2375     do
2376     {
2377         for (int i = 0; i < opt->data.getStudentNum(); i++)
2378         {
2379             if (!queryStudent(opt->data.getStudentNo()[i], false))
2380             {
2381                 Node<Student> *stu = new Node<Student>;
2382                 stu->data.setStudentNo(opt->data.getStudentNo()[i]);
2383                 stu->data.setName(opt->data.getStudentName()[i]);
2384                 stu->data.setLectureNum(stu->data.getLectureNum() + 1);
2385                 stu->data.getLectureName().push_back(opt->data.getName());
2386                 stu->data.getLectureType().push_back(OPTIONAL);
2387                 stu->data.getLectureCredit().push_back(opt->data.getCredit());
2388                 stu->data.getLectureScore().push_back(opt->data.getStudentScore()
[i]);
2389                 stu->data.getLectureGPA().push_back(opt->data.getStudentGPA()
[i]);
2390                 stu->data.getLecturePF().push_back(opt->data.getStudentPF()[i]);
2391                 stu->data.setLecturePFNum(stu->data.getLecturePFNum() + (opt-
>data.getStudentPF()[i] ? 1 : 0));
2392                 stu->data.setTotalScore(stu->data.getTotalScore() + opt-
>data.getStudentScore()[i] * (!opt->data.getStudentPF()[i] ? opt->data.getCredit() :
0));
2393                 stu->data.setTotalCredit(stu->data.getTotalCredit() + opt-
>data.getCredit());
2394                 stu->data.setTotalPFCredit(stu->data.getTotalPFCredit() + (opt-
>data.getStudentPF()[i] ? opt->data.getCredit() : 0));

```

```

2395         stu->data.setTotalGPA(stu->data.getTotalGPA() + calculateGPA(opt-
>data.getStudentScore()[i]) * opt->data.getCredit());
2396         if (stu->data.getTotalCredit() - stu->data.getTotalPFCredit())
2397             stu->data.setAverageScore(stu->data.getTotalScore() / (stu-
>data.getTotalCredit() - stu->data.getTotalPFCredit()));
2398         if (stu->data.getTotalCredit() - stu->data.getTotalPFCredit())
2399             stu->data.setAverageGPA(stu->data.getTotalGPA() / (stu-
>data.getTotalCredit() - stu->data.getTotalPFCredit()));
2400         studentList.insert_end(stu);
2401     }
2402     else
2403     {
2404         Student *stu = findStudent(opt->data.getStudentNo()[i]);
2405         stu->setLectureNum(stu->getLectureNum() + 1);
2406         stu->getLectureName().push_back(opt->data.getName());
2407         stu->getLectureType().push_back(OPTIONAL);
2408         stu->getLectureCredit().push_back(opt->data.getCredit());
2409         stu->getLectureScore().push_back(opt->data.getStudentScore()[i]);
2410         stu->getLectureGPA().push_back(opt->data.getStudentGPA()[i]);
2411         stu->getLecturePF().push_back(opt->data.getStudentPF()[i]);
2412         stu->setLecturePFNum(stu->getLecturePFNum() + (opt-
>data.getStudentPF()[i] ? 1 : 0));
2413         stu->setTotalScore(stu->getTotalScore() + opt-
>data.getStudentScore()[i] * (!opt->data.getStudentPF()[i] ? opt->data.getCredit() :
0));
2414         stu->setTotalCredit(stu->getTotalCredit() + opt-
>data.getCredit());
2415         stu->setTotalPFCredit(stu->getTotalPFCredit() + (opt-
>data.getStudentPF()[i] ? opt->data.getCredit() : 0));
2416         stu->setTotalGPA(stu->getTotalGPA() + calculateGPA(opt-
>data.getStudentScore()[i]) * opt->data.getCredit());
2417         if (stu->getTotalCredit() - stu->getTotalPFCredit())
2418             stu->setAverageScore(stu->getTotalScore() / (stu-
>getTotalCredit() - stu->getTotalPFCredit()));
2419         if (stu->getTotalCredit() - stu->getTotalPFCredit())
2420             stu->setAverageGPA(stu->getTotalGPA() / (stu-
>getTotalCredit() - stu->getTotalPFCredit()));
2421     }
2422 }
2423     opt = optionalList.getNext();
2424 } while (opt != nullptr);
2425 }
2426 }
2427
2428 // 从学生更新课程信息
2429 void Database::updateLecture()
2430 {
2431     requiredList.makeEmpty();
2432     limitedList.makeEmpty();
2433     optionalList.makeEmpty();
2434     Node<Student> *stu = studentList.getFirst();
2435     if (stu != nullptr)

```

```

2436     {
2437         do
2438         {
2439             for (int i = 0; i < stu->data.getLectureNum(); i++)
2440             {
2441                 if (stu->data.getLectureType()[i] == REQUIRED)
2442                 {
2443                     if (!queryRequired(stu->data.getLectureName()[i], false))
2444                     {
2445                         Node<LectureRequired> *req = new Node<LectureRequired>;
2446                         req->data.setName(stu->data.getLectureName()[i]);
2447                         req->data.setCredit(stu->data.getLectureCredit()[i]);
2448                         req->data.setStudentNum(req->data.getStudentNum() + 1);
2449                         std::vector<int> studentNo = req->data.getStudentNo();
2450                         std::vector<std::string> studentName = req-
2451 >data.getStudentName();
2452                         std::vector<int> studentScore = req->data.getStudentScore();
2453                         std::vector<double> studentGPA = req->data.getStudentGPA();
2454                         studentNo.push_back(stu->data.getStudentNo());
2455                         studentName.push_back(stu->data.getName());
2456                         studentScore.push_back(stu->data.getLectureScore()[i]);
2457                         studentGPA.push_back(stu->data.getLectureGPA()[i]);
2458                         req->data.setStudentNo(studentNo);
2459                         req->data.setStudentName(studentName);
2460                         req->data.setStudentScore(studentScore);
2461                         req->data.setStudentGPA(studentGPA);
2462                         req->data.setTotalScore(req->data.getTotalScore() + stu-
2463 >data.getLectureScore()[i]);
2464                         req->data.setTotalGPA(req->data.getTotalGPA() + stu-
2465 >data.getLectureGPA()[i]);
2466                         req->data.setAverageScore(req->data.getTotalScore() / req-
2467 >data.getStudentNum());
2468                         req->data.setAverageGPA(req->data.getTotalGPA() / req-
2469 >data.getStudentNum());
2470                         requiredList.insert_end(req);
2471                     }
2472                 else
2473                 {
2474                     LectureRequired *req = findRequired(stu-
2475 >data.getLectureName()[i]);
2476                     req->setStudentNum(req->getStudentNum() + 1);
2477                     std::vector<int> studentNo = req->getStudentNo();
2478                     std::vector<std::string> studentName = req->getStudentName();
2479                     std::vector<int> studentScore = req->getStudentScore();
2480                     std::vector<double> studentGPA = req->getStudentGPA();
2481                     studentNo.push_back(stu->data.getStudentNo());
2482                     studentName.push_back(stu->data.getName());
2483                     studentScore.push_back(stu->data.getLectureScore()[i]);
2484                     studentGPA.push_back(stu->data.getLectureGPA()[i]);
2485                     req->setStudentNo(studentNo);
2486                     req->setStudentName(studentName);
2487                     req->setStudentScore(studentScore);

```

```

2482         req->setStudentGPA(studentGPA);
2483         req->setTotalScore(req->getTotalScore() + stu-
>data.getLectureScore()[i]);
2484         req->setTotalGPA(req->getTotalGPA() + stu-
>data.getLectureGPA()[i]);
2485         req->setAverageScore(req->getTotalScore() / req-
>getStudentNum());
2486         req->setAverageGPA(req->getTotalGPA() / req-
>getStudentNum());
2487     }
2488 }
2489 else if (stu->data.getLectureType()[i] == LIMITED)
2490 {
2491     if (!queryLimited(stu->data.getLectureName()[i], false))
2492     {
2493         Node<LectureLimited> *lim = new Node<LectureLimited>;
2494         lim->data.setName(stu->data.getLectureName()[i]);
2495         lim->data.setCredit(stu->data.getLectureCredit()[i]);
2496         lim->data.setStudentNum(lim->data.getStudentNum() + 1);
2497         std::vector<int> studentNo = lim->data.getStudentNo();
2498         std::vector<std::string> studentName = lim-
>data.getStudentName();
2499         std::vector<int> studentScore = lim->data.getStudentScore();
2500         std::vector<double> studentGPA = lim->data.getStudentGPA();
2501         studentNo.push_back(stu->data.getStudentNo());
2502         studentName.push_back(stu->data.getName());
2503         studentScore.push_back(stu->data.getLectureScore()[i]);
2504         studentGPA.push_back(stu->data.getLectureGPA()[i]);
2505         lim->data.setStudentNo(studentNo);
2506         lim->data.setStudentName(studentName);
2507         lim->data.setStudentScore(studentScore);
2508         lim->data.setStudentGPA(studentGPA);
2509         lim->data.setTotalScore(lim->data.getTotalScore() + stu-
>data.getLectureScore()[i]);
2510         lim->data.setTotalGPA(lim->data.getTotalGPA() + stu-
>data.getLectureGPA()[i]);
2511         lim->data.setAverageScore(lim->data.getTotalScore() / lim-
>data.getStudentNum());
2512         lim->data.setAverageGPA(lim->data.getTotalGPA() / lim-
>data.getStudentNum());
2513         limitedList.insert_end(lim);
2514     }
2515     else
2516     {
2517         LectureLimited *lim = findLimited(stu->data.getLectureName()
[i]);
2518         lim->setStudentNum(lim->getStudentNum() + 1);
2519         std::vector<int> studentNo = lim->getStudentNo();
2520         std::vector<std::string> studentName = lim->getStudentName();
2521         std::vector<int> studentScore = lim->getStudentScore();
2522         std::vector<double> studentGPA = lim->getStudentGPA();
2523         studentNo.push_back(stu->data.getStudentNo());

```

```

2524         studentName.push_back(stu->data.getName());
2525         studentScore.push_back(stu->data.getLectureScore()[i]);
2526         studentGPA.push_back(stu->data.getLectureGPA()[i]);
2527         lim->setStudentNo(studentNo);
2528         lim->setStudentName(studentName);
2529         lim->setStudentScore(studentScore);
2530         lim->setStudentGPA(studentGPA);
2531         lim->setTotalScore(lim->getTotalScore() + stu-
>data.getLectureScore()[i]);
2532         lim->setTotalGPA(lim->getTotalGPA() + stu-
>data.getLectureGPA()[i]);
2533         lim->setAverageScore(lim->getTotalScore() / lim-
>getStudentNum());
2534     }
2535 }
2536 else if (stu->data.getLectureType()[i] == OPTIONAL)
2537 {
2538     if (!queryOptional(stu->data.getLectureName()[i], false))
2539     {
2540         Node<LectureOptional> *opt = new Node<LectureOptional>;
2541         opt->data.setName(stu->data.getLectureName()[i]);
2542         opt->data.setCredit(stu->data.getLectureCredit()[i]);
2543         opt->data.setStudentNum(opt->data.getStudentNum() + 1);
2544         std::vector<int> studentNo = opt->data.getStudentNo();
2545         std::vector<std::string> studentName = opt-
>data.getStudentName();
2546         std::vector<int> studentScore = opt->data.getStudentScore();
2547         std::vector<double> studentGPA = opt->data.getStudentGPA();
2548         std::vector<int> studentPF = opt->data.getStudentPF();
2549         studentNo.push_back(stu->data.getStudentNo());
2550         studentName.push_back(stu->data.getName());
2551         studentScore.push_back(stu->data.getLectureScore()[i]);
2552         studentGPA.push_back(stu->data.getLectureGPA()[i]);
2553         studentPF.push_back(stu->data.getLecturePF()[i]);
2554         opt->data.setStudentNo(studentNo);
2555         opt->data.setStudentName(studentName);
2556         opt->data.setStudentScore(studentScore);
2557         opt->data.setStudentGPA(studentGPA);
2558         opt->data.setStudentPF(studentPF);
2559         opt->data.setStudentPFNum(opt->data.getStudentPFNum() + stu-
>data.getLecturePF()[i]);
2560         opt->data.setTotalScore(opt->data.getTotalScore() + (!stu-
>data.getLecturePF()[i] ? stu->data.getLectureScore()[i] : 0));
2561         opt->data.setTotalGPA(opt->data.getTotalGPA() + (!stu-
>data.getLecturePF()[i] ? stu->data.getLectureGPA()[i] : 0));
2562         if (opt->data.getStudentNum() - opt->data.getStudentPFNum())
2563             opt->data.setAverageScore(opt->data.getTotalScore() /
(opt->data.getStudentNum() - opt->data.getStudentPFNum()));
2564         if (opt->data.getStudentNum() - opt->data.getStudentPFNum())
2565             opt->data.setAverageGPA(opt->data.getTotalGPA() / (opt-
>data.getStudentNum() - opt->data.getStudentPFNum()));
2566         optionalList.insert_end(opt);

```

```

2567         }
2568         else
2569         {
2570             LectureOptional *opt = findOptional(stu-
>data.getLectureName()[i]);
2571             opt->setStudentNum(opt->getStudentNum() + 1);
2572             std::vector<int> studentNo = opt->getStudentNo();
2573             std::vector<std::string> studentName = opt->getStudentName();
2574             std::vector<int> studentScore = opt->getStudentScore();
2575             std::vector<double> studentGPA = opt->getStudentGPA();
2576             std::vector<int> studentPF = opt->getStudentPF();
2577             studentNo.push_back(stu->data.getStudentNo());
2578             studentName.push_back(stu->data.getName());
2579             studentScore.push_back(stu->data.getLectureScore()[i]);
2580             studentGPA.push_back(stu->data.getLectureGPA()[i]);
2581             studentPF.push_back(stu->data.getLecturePF()[i]);
2582             opt->setStudentNo(studentNo);
2583             opt->setStudentName(studentName);
2584             opt->setStudentScore(studentScore);
2585             opt->setStudentGPA(studentGPA);
2586             opt->setStudentPF(studentPF);
2587             opt->setStudentPFNum(opt->getStudentPFNum() + stu-
>data.getLecturePF()[i]);
2588             opt->setTotalScore(opt->getTotalScore() + (!stu-
>data.getLecturePF()[i] ? stu->data.getLectureScore()[i] : 0));
2589             opt->setTotalGPA(opt->getTotalGPA() + (!stu-
>data.getLecturePF()[i] ? stu->data.getLectureGPA()[i] : 0));
2590             if (opt->getStudentNum() - opt->getStudentPFNum())
2591                 opt->setAverageScore(opt->getTotalScore() / (opt-
>getStudentNum() - opt->getStudentPFNum()));
2592             if (opt->getStudentNum() - opt->getStudentPFNum())
2593                 opt->setAverageGPA(opt->getTotalGPA() / (opt-
>getStudentNum() - opt->getStudentPFNum()));
2594         }
2595     }
2596 }
2597 stu = studentList.getNext();
2598 } while (stu != nullptr);
2599 }
2600 }
2601
2602 // 登录
2603 Account *Database::login(std::string username, std::string password)
2604 {
2605     Node<User> *user = userList.getFirst();
2606     while (user != nullptr)
2607     {
2608         if (user->data.getName() == username && user->data.getPassword() == password)
2609             return &user->data;
2610         user = userList.getNext();
2611     }
2612     Node<Admin> *admin = adminList.getFirst();

```



```

2613     while (admin != nullptr)
2614     {
2615         if (admin->data.getName() == username && admin->data.getPassword() ==
password)
2616             return &admin->data;
2617         admin = adminList.getNext();
2618     }
2619     return nullptr;
2620 }
2621
2622 // 注册
2623 Account *Database::registerUser(std::string username, std::string password, int
permission)
2624 {
2625     if (permission == 1)
2626     {
2627         Node<User> *acc = new Node<User>;
2628         acc->data.setName(username);
2629         acc->data.setPassword(password);
2630         userList.insert_end(acc);
2631         return &acc->data;
2632     }
2633     else if (permission == 2)
2634     {
2635         Node<Admin> *acc = new Node<Admin>;
2636         acc->data.setName(username);
2637         acc->data.setPassword(password);
2638         adminList.insert_end(acc);
2639         return &acc->data;
2640     }
2641     else
2642     {
2643         return nullptr;
2644     }
2645 }
2646
2647 // 加载账号
2648 void Database::loadAccount(const std::string &userFilename, const std::string
&adminFilename)
2649 {
2650     userList.makeEmpty();
2651     adminList.makeEmpty();
2652     encrypt(userFilename);
2653     encrypt(adminFilename);
2654     std::ifstream in;
2655     in.open(userFilename.c_str(), std::ios::in);
2656     Node<User> *user;
2657     int tmpPermission = 0;
2658     // 打开文件成功
2659     if (in)
2660     {
2661         in.seekg(0, std::ios::end);

```

```

2662     int fileSize = in.tellg();
2663     in.seekg(std::ios::beg);
2664     while (fileSize - in.tellg() > 2)
2665     {
2666         user = new Node<User>;
2667         if (in >> user->data)
2668         {
2669             userList.insert_end(user);
2670         }
2671         else
2672         {
2673             FileException e(userFilename, "operate", "read");
2674             throw e;
2675         }
2676     }
2677 }
2678 else
2679 {
2680     FileException e(userFilename, "open", "read");
2681     throw e;
2682 }
2683 in.close();
2684 in.open(adminFilename.c_str(), std::ios::in);
2685 Node<Admin> *admin;
2686 // 打开文件成功
2687 if (in)
2688 {
2689     in.seekg(0, std::ios::end);
2690     int fileSize = in.tellg();
2691     in.seekg(std::ios::beg);
2692     while (fileSize - in.tellg() > 2)
2693     {
2694         admin = new Node<Admin>;
2695         if (in >> admin->data)
2696         {
2697             adminList.insert_end(admin);
2698         }
2699         else
2700         {
2701             FileException e(adminFilename, "operate", "read");
2702             throw e;
2703         }
2704     }
2705 }
2706 else
2707 {
2708     FileException e(adminFilename, "open", "read");
2709     throw e;
2710 }
2711 encrypt(userFilename);
2712 encrypt(adminFilename);
2713 }

```

```

2714
2715 // 保存账号
2716 void Database::saveAccount(const std::string &userFilename, const std::string
&adminFilename)
2717 {
2718     encrypt(userFilename);
2719     encrypt(adminFilename);
2720     std::ofstream out;
2721     out.open(userFilename.c_str(), std::ios::out);
2722     Node<User> *user = userList.getFirst();
2723     // 打开文件成功
2724     if (out)
2725     {
2726         while (user != nullptr)
2727         {
2728             if (!(out << user->data))
2729             {
2730                 FileException e(userFilename, "operate", "write");
2731                 throw e;
2732             }
2733             user = userList.getNext();
2734         }
2735     }
2736     else
2737     {
2738         FileException e(userFilename, "open", "write");
2739         throw e;
2740     }
2741     out.close();
2742     out.open(adminFilename.c_str(), std::ios::out);
2743     Node<Admin> *admin = adminList.getFirst();
2744     // 打开文件成功
2745     if (out)
2746     {
2747         while (admin != nullptr)
2748         {
2749             if (!(out << admin->data))
2750             {
2751                 FileException e(adminFilename, "operate", "write");
2752                 throw e;
2753             }
2754             admin = adminList.getNext();
2755         }
2756     }
2757     else
2758     {
2759         FileException e(adminFilename, "open", "write");
2760         throw e;
2761     }
2762     encrypt(userFilename);
2763     encrypt(adminFilename);
2764 }

```

```

2765
2766 // 查询账号
2767 int Database::queryAccount(const std::string& username)
2768 {
2769     int count = 0;
2770     Node<User>*user = userList.getFirst();
2771     while (user != nullptr)
2772     {
2773         if (user->data.getName() == username)
2774             count++;
2775         user = userList.getNext();
2776     }
2777     Node<Admin>*admin = adminList.getFirst();
2778     while (admin != nullptr)
2779     {
2780         if (admin->data.getName() == username)
2781             count++;
2782         admin = adminList.getNext();
2783     }
2784     return count;
2785 }
2786

```

## info.cpp

```

1  #include "commonheader.h"
2
3  // 构造函数
4  Info::Info()
5  {
6      name = "DefaultName";
7      uid = currentUid++;
8  }
9
10 // 构造函数
11 Info::Info(std::string inputName)
12 {
13     name = inputName;
14     uid = currentUid++;
15 }
16
17 // 析构函数
18 Info::~Info() {}
19
20 // 获取名称
21 std::string Info::getName()
22 {
23     return name;
24 }
25
26 // 获取唯一标识符

```

```

27 int Info::getUid()
28 {
29     return uid;
30 }
31
32 // 获取是否处于调试模式
33 bool Info::isDebugMode()
34 {
35     return DebugMode;
36 }
37
38 // 设置名称
39 void Info::setName(std::string inputName)
40 {
41     name = inputName;
42 }
43

```

## lecture.cpp

```

1  #include "commonheader.h"
2
3  // 构造函数
4  Lecture::Lecture()
5  {
6      credit = 0;
7      studentNo.clear();
8      studentName.clear();
9      studentScore.clear();
10     studentGPA.clear();
11     studentNum = 0;
12     totalScore = 0;
13     totalGPA = 0;
14     averageScore = 0;
15     averageGPA = 0;
16 }
17
18 // 析构函数
19 Lecture::~Lecture() {}
20
21 // 获取学分
22 int Lecture::getCredit()
23 {
24     return credit;
25 }
26
27 // 获取学号
28 std::vector<int> Lecture::getStudentNo()
29 {
30     return studentNo;
31 }

```

```
32
33 // 获取姓名
34 std::vector<std::string> Lecture::getStudentName()
35 {
36     return studentName;
37 }
38
39 // 获取成绩
40 std::vector<int> Lecture::getStudentScore()
41 {
42     return studentScore;
43 }
44
45 // 获取绩点
46 std::vector<double> Lecture::getStudentGPA()
47 {
48     return studentGPA;
49 }
50
51 // 获取学生人数
52 int Lecture::getStudentNum()
53 {
54     return studentNum;
55 }
56
57 // 获取总分
58 int Lecture::getTotalScore()
59 {
60     return totalScore;
61 }
62
63 // 获取总绩点
64 double Lecture::getTotalGPA()
65 {
66     return totalGPA;
67 }
68
69 // 获取平均分
70 int Lecture::getAverageScore()
71 {
72     return averageScore;
73 }
74
75 // 获取平均绩点
76 double Lecture::getAverageGPA()
77 {
78     return averageGPA;
79 }
80
81 // 更新课程信息
82 void Lecture::updateInfo(Database &database)
83 {
```

```

84     bool validName = false, validCredit = false;
85     std::string inputName;
86     int inputCredit;
87     std::string oldName = name;
88
89     do
90     {
91         std::cout << "请输入名称: ";
92         std::cin >> inputName;
93         if (database.queryLecture(inputName, false) && inputName != oldName)
94         {
95             std::cout << "该名称已存在, 请重新输入! " << std::endl;
96         }
97         else
98         {
99             name = inputName;
100             validName = true;
101         }
102     } while (!validName);
103
104     do
105     {
106         std::cout << "请输入学分: ";
107         std::cin >> inputCredit;
108         if (inputCredit < 0)
109         {
110             std::cout << "学分不能为负数, 请重新输入! " << std::endl;
111         }
112         else
113         {
114             credit = inputCredit;
115             validCredit = true;
116         }
117     } while (!validCredit);
118
119     // TODO: update database
120 }
121
122 // 设置名称
123 void Lecture::setName(std::string inputName)
124 {
125     name = inputName;
126 }
127
128 // 设置学分
129 void Lecture::setCredit(int inputCredit)
130 {
131     credit = inputCredit;
132 }
133
134 // 设置学号
135 void Lecture::setStudentNo(std::vector<int> inputStudentNo)

```

```
136 {
137     studentNo = inputStudentNo;
138 }
139
140 // 设置姓名
141 void Lecture::setStudentName(std::vector<std::string> inputStudentName)
142 {
143     studentName = inputStudentName;
144 }
145
146 // 设置成绩
147 void Lecture::setStudentScore(std::vector<int> inputStudentScore)
148 {
149     studentScore = inputStudentScore;
150 }
151
152 // 设置绩点
153 void Lecture::setStudentGPA(std::vector<double> inputStudentGPA)
154 {
155     studentGPA = inputStudentGPA;
156 }
157
158 // 设置学生人数
159 void Lecture::setStudentNum(int inputStudentNum)
160 {
161     studentNum = inputStudentNum;
162 }
163
164 // 设置总分
165 void Lecture::setTotalScore(int inputTotalScore)
166 {
167     totalScore = inputTotalScore;
168 }
169
170 // 设置总绩点
171 void Lecture::setTotalGPA(double inputTotalGPA)
172 {
173     totalGPA = inputTotalGPA;
174 }
175
176 // 设置平均分
177 void Lecture::setAverageScore(int inputAverageScore)
178 {
179     averageScore = inputAverageScore;
180 }
181
182 // 设置平均绩点
183 void Lecture::setAverageGPA(double inputAverageGPA)
184 {
185     averageGPA = inputAverageGPA;
186 }
187
```



```

188 // 添加学生
189 void Lecture::addStudent(int inputStudentNo, std::string inputStudentName, int
inputStudentScore, double inputStudentGPA)
190 {
191     studentNo.push_back(inputStudentNo);
192     studentName.push_back(inputStudentName);
193     studentScore.push_back(inputStudentScore);
194     studentGPA.push_back(inputStudentGPA);
195     studentNum++;
196     totalScore += inputStudentScore;
197     totalGPA += inputStudentGPA;
198     averageScore = totalScore / studentNum;
199     averageGPA = totalGPA / studentNum;
200 }
201
202 // 简略打印课程信息
203 void Lecture::printInfo(int widthName)
204 {
205     std::cout << std::setw(widthName) << name
206         << std::setw(5) << "通用"
207         << std::setw(5) << credit
208         << std::setw(5) << studentNum
209         << std::setw(5) << std::setprecision(3) << averageScore
210         << std::setw(5) << std::setprecision(3) << averageGPA << std::endl;
211 }
212
213 // 详细打印课程信息
214 void Lecture::printLectureInfo()
215 {
216     std::cout << std::endl;
217     if (isDebugMode())
218         std::cout << "    UID: " << uid << std::endl;
219     std::cout << "课程名称: " << name << std::endl;
220     std::cout << "课程学分: " << credit << std::endl;
221     std::cout << "学生人数: " << studentNum << std::endl;
222     std::cout << "平均分数: " << averageScore << std::endl;
223     std::cout << "平均绩点: " << averageGPA << std::endl;
224     int widthStudentNo = 5, widthStudentName = 5, widthStudentScore = 5,
widthStudentGPA = 5;
225     for (int i = 0; i < studentNum; i++)
226     {
227         if (std::to_string(studentNo[i]).length() >= widthStudentNo)
228         {
229             widthStudentNo = std::to_string(studentNo[i]).length() + 1;
230         }
231         if (studentName[i].length() >= widthStudentName)
232         {
233             widthStudentName = studentName[i].length() + 1;
234         }
235     }
236     std::cout << std::setw(widthStudentNo) << "学号"
237         << std::setw(widthStudentName) << "姓名"

```

```

238         << std::setw(5) << "成绩"
239         << std::setw(5) << "绩点" << std::endl;
240     for (int i = 0; i < studentNum; i++)
241     {
242         std::cout << std::setw(widthStudentNo) << studentNo[i]
243                 << std::setw(widthStudentName) << studentName[i]
244                 << std::setw(5) << studentScore[i]
245                 << std::setw(5) << studentGPA[i] << std::endl;
246     }
247 }
248
249 // 流输入操作符重载函数
250 std::istream &operator>>(std::istream &is, Lecture &lecture)
251 {
252     is >> lecture.name >> lecture.credit >> lecture.studentNum >> lecture.totalScore
253     >> lecture.totalGPA >> lecture.averageScore >> lecture.averageGPA;
254     lecture.studentNo.resize(lecture.studentNum);
255     lecture.studentName.resize(lecture.studentNum);
256     lecture.studentScore.resize(lecture.studentNum);
257     lecture.studentGPA.resize(lecture.studentNum);
258     for (int i = 0; i < lecture.studentNum; i++)
259     {
260         is >> lecture.studentNo[i] >> lecture.studentName[i] >>
261         lecture.studentScore[i] >> lecture.studentGPA[i];
262     }
263     return is;
264 }
265
266 // 流输出操作符重载函数
267 std::ostream &operator<<(std::ostream &os, Lecture &lecture)
268 {
269     os << lecture.name << "\t"
270     << lecture.credit << "\t"
271     << lecture.studentNum << "\t"
272     << lecture.totalScore << "\t"
273     << lecture.totalGPA << "\t"
274     << lecture.averageScore << "\t"
275     << lecture.averageGPA << std::endl;
276     for (int i = 0; i < lecture.studentNum; i++)
277     {
278         os << lecture.studentNo[i] << "\t"
279         << lecture.studentName[i] << "\t"
280         << lecture.studentScore[i] << "\t"
281         << lecture.studentGPA[i] << std::endl;
282     }
283     return os;
284 }

```

## lecture\_limited.cpp

```
1  #include "commonheader.h"
2
3  // 构造函数
4  LectureLimited::LectureLimited()
5  {
6  }
7
8  // 析构函数
9  LectureLimited::~~LectureLimited()
10 {
11 }
12
13 // 更新课程信息
14 void LectureLimited::updateInfo(Database &database)
15 {
16     Lecture::updateInfo(database);
17 }
18
19 // 简略打印课程信息
20 void LectureLimited::printInfo(int widthName)
21 {
22     std::cout << std::setw(widthName) << name
23               << std::setw(5) << "限选课"
24               << std::setw(5) << credit
25               << std::setw(5) << studentNum
26               << std::setw(5) << std::setprecision(3) << averageScore
27               << std::setw(5) << std::setprecision(3) << averageGPA << std::endl;
28 }
29
30 // 详细打印课程信息
31 void LectureLimited::printLectureInfo()
32 {
33     if (isDebugMode())
34         std::cout << "UID: " << uid << std::endl;
35     std::cout << "课程名称: " << name << std::endl
36               << "课程类型: 限选课" << std::endl
37               << "课程学分: " << credit << std::endl
38               << "学生人数: " << studentNum << std::endl
39               << "平均分数: " << averageScore << std::endl
40               << "平均绩点: " << averageGPA << std::endl;
41     int widthStudentNo = 5, widthStudentName = 5;
42     for (int i = 0; i < studentNum; i++)
43     {
44         if (std::to_string(studentNo[i]).length() >= widthStudentNo)
45         {
46             widthStudentNo = std::to_string(studentNo[i]).length() + 1;
47         }
48         if (studentName[i].length() >= widthStudentName)
49         {
```

```

50         widthStudentName = studentName[i].length() + 1;
51     }
52 }
53 std::cout << std::setw(widthStudentNo) << "学号"
54         << std::setw(widthStudentName) << "姓名"
55         << std::setw(5) << "成绩"
56         << std::setw(5) << "绩点" << std::endl;
57 for (int i = 0; i < studentNum; i++)
58 {
59     std::cout << std::setw(widthStudentNo) << studentNo[i]
60             << std::setw(widthStudentName) << studentName[i]
61             << std::setw(5) << studentScore[i]
62             << std::setw(5) << studentGPA[i] << std::endl;
63 }
64 }
65
66 // 流输入操作符重载函数
67 std::istream &operator>>(std::istream &is, LectureLimited &lecture)
68 {
69     is >> lecture.name >> lecture.credit >> lecture.studentNum >> lecture.totalScore
70     >> lecture.totalGPA >> lecture.averageScore >> lecture.averageGPA;
71     lecture.studentNo.resize(lecture.studentNum);
72     lecture.studentName.resize(lecture.studentNum);
73     lecture.studentScore.resize(lecture.studentNum);
74     lecture.studentGPA.resize(lecture.studentNum);
75     for (int i = 0; i < lecture.studentNum; i++)
76     {
77         is >> lecture.studentNo[i] >> lecture.studentName[i] >>
78         lecture.studentScore[i] >> lecture.studentGPA[i];
79     }
80     return is;
81 }
82
83 // 流输出操作符重载函数
84 std::ostream &operator<<(std::ostream &os, LectureLimited &lecture)
85 {
86     os << lecture.name << "\t"
87     << lecture.credit << "\t"
88     << lecture.studentNum << "\t"
89     << lecture.totalScore << "\t"
90     << lecture.totalGPA << "\t"
91     << lecture.averageScore << "\t"
92     << lecture.averageGPA << std::endl;
93     for (int i = 0; i < lecture.studentNum; i++)
94     {
95         os << lecture.studentNo[i] << "\t"
96         << lecture.studentName[i] << "\t"
97         << lecture.studentScore[i] << "\t"
98         << lecture.studentGPA[i] << std::endl;
99     }
100     return os;
101 }

```

**limited\_optional.cpp**

```
1  #include "commonheader.h"
2
3  // 构造函数
4  LectureOptional::LectureOptional()
5  {
6      studentPF.clear();
7      studentPFNum = 0;
8  }
9
10 // 析构函数
11 LectureOptional::~LectureOptional()
12 {
13 }
14
15 // 获取学生计PF情况
16 std::vector<int> LectureOptional::getStudentPF()
17 {
18     return studentPF;
19 }
20
21 // 获取学生计PF人数
22 int LectureOptional::getStudentPFNum()
23 {
24     return studentPFNum;
25 }
26
27 // 设置学生计PF情况
28 void LectureOptional::setStudentPF(std::vector<int> inputStudentPF)
29 {
30     studentPF = inputStudentPF;
31 }
32
33 // 设置学生计PF人数
34 void LectureOptional::setStudentPFNum(int inputStudentPFNum)
35 {
36     studentPFNum = inputStudentPFNum;
37 }
38
39 // 添加学生
40 void LectureOptional::addStudent(int inputStudentNo, std::string inputStudentName, int
inputStudentScore, double inputStudentGPA, bool inputStudentPF)
41 {
42     studentNo.push_back(inputStudentNo);
43     studentName.push_back(inputStudentName);
44     studentScore.push_back(inputStudentScore);
45     studentGPA.push_back(inputStudentGPA);
46     studentPF.push_back(inputStudentPF);
```

```

47     studentNum++;
48     if (!inputStudentPF)
49     {
50         totalScore += inputStudentScore;
51         totalGPA += inputStudentGPA;
52     }
53     else
54     {
55         studentPFNum++;
56     }
57     averageScore = totalScore / (studentNum - studentPFNum);
58     averageGPA = totalGPA / (studentNum - studentPFNum);
59 }
60
61 // 更新课程信息
62 void LectureOptional::updateInfo(Database &database)
63 {
64     Lecture::updateInfo(database);
65 }
66
67 // 简略打印课程信息
68 void LectureOptional::printInfo(int widthName)
69 {
70     std::cout << std::setw(widthName) << name
71         << std::setw(5) << "任选"
72         << std::setw(5) << credit
73         << std::setw(5) << studentNum
74         << std::setw(5) << std::setprecision(3) << averageScore
75         << std::setw(5) << std::setprecision(3) << averageGPA << std::endl;
76 }
77
78 // 详细打印课程信息
79 void LectureOptional::printLectureInfo()
80 {
81     if (isDebugMode())
82         std::cout << "UID: " << uid << std::endl;
83     std::cout << "课程名称: " << name << std::endl;
84     std::cout << "课程类型: 任选课" << std::endl;
85     std::cout << "课程学分: " << credit << std::endl;
86     std::cout << "学生人数: " << studentNum << std::endl;
87     std::cout << "平均分数: " << averageScore << std::endl;
88     std::cout << "平均绩点: " << averageGPA << std::endl;
89     int widthStudentNo = 5, widthStudentName = 5;
90     for (int i = 0; i < studentNum; i++)
91     {
92         if (std::to_string(studentNo[i]).length() >= widthStudentNo)
93         {
94             widthStudentNo = std::to_string(studentNo[i]).length() + 1;
95         }
96         if (studentName[i].length() >= widthStudentName)
97         {
98             widthStudentName = studentName[i].length() + 1;

```

```

99     }
100 }
101 std::cout << std::setw(widthStudentNo) << "学号"
102         << std::setw(widthStudentName) << "姓名"
103         << std::setw(5) << "成绩"
104         << std::setw(5) << "绩点" << std::endl;
105 for (int i = 0; i < studentNum; i++)
106 {
107     std::cout << std::setw(widthStudentNo) << studentNo[i]
108             << std::setw(widthStudentName) << studentName[i];
109     std::cout.width(5);
110     if (studentPF[i])
111         std::cout << "N/A";
112     else
113         std::cout << studentScore[i];
114     std::cout.width(5);
115     if (studentPF[i])
116         std::cout << (studentGPA[i] ? "P" : "F") << std::endl;
117     else
118         std::cout << studentGPA[i] << std::endl;
119 }
120 }
121
122 // 流输入操作符重载函数
123 std::istream &operator>>(std::istream &is, LectureOptional &lecture)
124 {
125     is >> lecture.name >> lecture.credit >> lecture.studentNum >> lecture.studentPFNum
126     >> lecture.totalScore >> lecture.totalGPA >> lecture.averageScore >>
127     lecture.averageGPA;
128     lecture.studentNo.resize(lecture.studentNum);
129     lecture.studentName.resize(lecture.studentNum);
130     lecture.studentScore.resize(lecture.studentNum);
131     lecture.studentGPA.resize(lecture.studentNum);
132     lecture.studentPF.resize(lecture.studentNum);
133     for (int i = 0; i < lecture.studentNum; i++)
134     {
135         is >> lecture.studentNo[i] >> lecture.studentName[i] >>
136         lecture.studentScore[i] >> lecture.studentGPA[i] >> lecture.studentPF[i];
137     }
138     return is;
139 }
140
141 // 流输出操作符重载函数
142 std::ostream &operator<<(std::ostream &os, LectureOptional &lecture)
143 {
144     os << lecture.name << "\t"
145         << lecture.credit << "\t"
146         << lecture.studentNum << "\t"
147         << lecture.studentPFNum << "\t"
148         << lecture.totalScore << "\t"
149         << lecture.totalGPA << "\t"
150         << lecture.averageScore << "\t"

```

```

148     << lecture.averageGPA << std::endl;
149     for (int i = 0; i < lecture.studentNum; i++)
150     {
151         os << lecture.studentNo[i] << "\t"
152         << lecture.studentName[i] << "\t"
153         << lecture.studentScore[i] << "\t"
154         << lecture.studentGPA[i] << "\t"
155         << lecture.studentPF[i] << std::endl;
156     }
157     return os;
158 }
159

```

## lecture\_required.cpp

```

1  #include "commonheader.h"
2
3  // 构造函数
4  LectureRequired::LectureRequired()
5  {
6  }
7
8  // 析构函数
9  LectureRequired::~LectureRequired()
10 {
11 }
12
13 // 更新课程信息
14 void LectureRequired::updateInfo(Database &database)
15 {
16     Lecture::updateInfo(database);
17 }
18
19 // 简略打印课程信息
20 void LectureRequired::printInfo(int widthName)
21 {
22     std::cout << std::setw(widthName) << name
23     << std::setw(5) << "必选"
24     << std::setw(5) << credit
25     << std::setw(5) << studentNum
26     << std::setw(5) << std::setprecision(3) << averageScore
27     << std::setw(5) << std::setprecision(3) << averageGPA << std::endl;
28 }
29
30 // 详细打印课程信息
31 void LectureRequired::printLectureInfo()
32 {
33     if (isDebugMode())
34         std::cout << "UID: " << uid << std::endl;
35     std::cout << "课程名称: " << name << std::endl;
36     std::cout << "课程类型: 必选课" << std::endl;

```



```

37     std::cout << "课程学分: " << credit << std::endl;
38     std::cout << "学生人数: " << studentNum << std::endl;
39     std::cout << "平均分数: " << averageScore << std::endl;
40     std::cout << "平均绩点: " << averageGPA << std::endl;
41     int widthStudentNo = 5, widthStudentName = 5;
42     for (int i = 0; i < studentNum; i++)
43     {
44         if (std::to_string(studentNo[i]).length() >= widthStudentNo)
45         {
46             widthStudentNo = std::to_string(studentNo[i]).length() + 1;
47         }
48         if (studentName[i].length() >= widthStudentName)
49         {
50             widthStudentName = studentName[i].length() + 1;
51         }
52     }
53     std::cout << std::setw(widthStudentNo) << "学号"
54               << std::setw(widthStudentName) << "姓名"
55               << std::setw(5) << "成绩"
56               << std::setw(5) << "绩点" << std::endl;
57     for (int i = 0; i < studentNum; i++)
58     {
59         std::cout << std::setw(widthStudentNo) << studentNo[i]
60               << std::setw(widthStudentName) << studentName[i]
61               << std::setw(5) << studentScore[i]
62               << std::setw(5) << studentGPA[i] << std::endl;
63     }
64 }
65
66 // 流输入操作符重载函数
67 std::istream &operator>>(std::istream &is, LectureRequired &lecture)
68 {
69     is >> lecture.name >> lecture.credit >> lecture.studentNum >> lecture.totalScore
70     >> lecture.totalGPA >> lecture.averageScore >> lecture.averageGPA;
71     lecture.studentNo.resize(lecture.studentNum);
72     lecture.studentName.resize(lecture.studentNum);
73     lecture.studentScore.resize(lecture.studentNum);
74     lecture.studentGPA.resize(lecture.studentNum);
75     for (int i = 0; i < lecture.studentNum; i++)
76     {
77         is >> lecture.studentNo[i] >> lecture.studentName[i] >>
78         lecture.studentScore[i] >> lecture.studentGPA[i];
79     }
80     return is;
81 }
82
83 // 流输出操作符重载函数
84 std::ostream &operator<<(std::ostream &os, LectureRequired &lecture)
85 {
86     os << lecture.name << "\t"
87       << lecture.credit << "\t"
88       << lecture.studentNum << "\t"

```

```

87         << lecture.totalScore << "\t"
88         << lecture.totalGPA << "\t"
89         << lecture.averageScore << "\t"
90         << lecture.averageGPA << std::endl;
91     for (int i = 0; i < lecture.studentNum; i++)
92     {
93         os << lecture.studentNo[i] << "\t"
94         << lecture.studentName[i] << "\t"
95         << lecture.studentScore[i] << "\t"
96         << lecture.studentGPA[i] << std::endl;
97     }
98     return os;
99 }
100

```

### main.cpp

```

1  #include "commonheader.h"
2
3  int Info::currentUid = 0;
4  bool Info::DebugMode = false;
5  Account *UserInterface::currentUser = nullptr;
6
7  int main()
8  {
9      UserInterface ui;
10     while (ui.run())
11         ;
12     return 0;
13 }
14

```

### student.cpp

```

1  #include "commonheader.h"
2
3  // 构造函数
4  Student::Student()
5  {
6      studentNo = 0;
7      lectureName.clear();
8      lectureScore.clear();
9      lectureCredit.clear();
10     lectureGPA.clear();
11     lectureNum = 0;
12     lecturePFNum = 0;
13     totalScore = 0;
14     totalCredit = 0;
15     totalPFCredit = 0;
16     totalGPA = 0;
17     averageScore = 0;

```

```
18     averageGPA = 0;
19 }
20
21 // 析构函数
22 Student::~~Student() {}
23
24 // 获取学号
25 int Student::getStudentNo()
26 {
27     return studentNo;
28 }
29
30 // 获取姓名
31 std::string Student::getName()
32 {
33     return name;
34 }
35
36 // 获取各科名称
37 std::vector<std::string> Student::getLectureName()
38 {
39     return lectureName;
40 }
41
42 // 获取各科类型
43 std::vector<LectureType> Student::getLectureType()
44 {
45     return lectureType;
46 }
47
48 // 获取各科成绩
49 std::vector<int> Student::getLectureScore()
50 {
51     return lectureScore;
52 }
53
54 // 获取各科学分
55 std::vector<int> Student::getLectureCredit()
56 {
57     return lectureCredit;
58 }
59
60 // 获取各科学分绩点
61 std::vector<double> Student::getLectureGPA()
62 {
63     return lectureGPA;
64 }
65
66 // 获取各科是否计PF
67 std::vector<int> Student::getLecturePF()
68 {
69     return lecturePF;
```

```
70 }
71
72 // 获取课程数
73 int Student::getLectureNum()
74 {
75     return lectureNum;
76 }
77
78 // 获取PF课程数
79 int Student::getLecturePFNum()
80 {
81     return lecturePFNum;
82 }
83
84 // 获取总成绩
85 int Student::getTotalScore()
86 {
87     return totalScore;
88 }
89
90 // 获取总学分
91 int Student::getTotalCredit()
92 {
93     return totalCredit;
94 }
95
96 // 获取PF学分
97 int Student::getTotalPFCredit()
98 {
99     return totalPFCredit;
100 }
101
102 // 获取总学分绩点
103 double Student::getTotalGPA()
104 {
105     return totalGPA;
106 }
107
108 // 获取平均成绩
109 double Student::getAverageScore()
110 {
111     return averageScore;
112 }
113
114 // 获取平均学分绩点
115 double Student::getAverageGPA()
116 {
117     return averageGPA;
118 }
119
120 // 修改学生信息
121 void Student::updateInfo(Database &database)
```

```

122 {
123     int oldStudentNo = studentNo;
124     std::string oldName = name;
125     bool validStudentNo = false, validName = false, validLecture = false, newLecture =
false, moreLecture = false;
126     int existLecture = 0;
127     int inputStudentNo;
128     std::string inputName;
129     std::string tmpLectureName;
130     int tmpLectureType;
131     int tmpLectureScore;
132     int tmpLectureCredit;
133     double tmpLectureGPA;
134     bool tmpLecturePF = false;
135     LectureType destLectureType;
136
137     do
138     {
139         std::cout << "请输入学号: ";
140         std::cin >> inputStudentNo;
141         if (database.queryStudent(inputStudentNo, false) && oldStudentNo !=
inputStudentNo)
142         {
143             std::cout << "该学号已存在, 请重新输入! " << std::endl;
144         }
145         else
146         {
147             studentNo = inputStudentNo;
148             validStudentNo = true;
149         }
150     } while (!validStudentNo);
151
152     do
153     {
154         std::cout << "请输入姓名: ";
155         std::cin >> inputName;
156         if (database.queryStudent(inputName, false) && oldName != inputName)
157         {
158             std::cout << "该姓名已存在, 请重新输入! " << std::endl;
159         }
160         else
161         {
162             name = inputName;
163             validName = true;
164         }
165     } while (!validName);
166
167     do
168     {
169         validLecture = newLecture = moreLecture = tmpLecturePF = false;
170         existLecture = 0;
171         std::cout << "请输入课程名称: ";

```

```

172     std::cin >> tmpLectureName;
173     if (!lectureName.empty())
174     {
175         for (int i = 0; i < lectureName.size(); i++)
176         {
177             if (lectureName[i] == tmpLectureName)
178             {
179                 existLecture = i + 1;
180                 break;
181             }
182         }
183     }
184     if (existLecture)
185     {
186         std::cout << "该学生已有该课程的成绩记录。是否覆盖? [Y/N]" << std::endl;
187         int cover;
188         cover = _getch();
189         if (cover == 'Y' || cover == 'y')
190         {
191             destLectureType = (database.queryRequired(tmpLectureName, false) ?
REQUIRED : (database.queryLimited(tmpLectureName, false) ? LIMITED : OPTIONAL));
192             std::cout << "请输入课程成绩 (0~100) : ";
193             std::cin >> tmpLectureScore;
194             if (!std::cin)
195             {
196                 std::cout << "输入错误! " << std::endl;
197                 std::cin.clear();
198                 std::cin.ignore();
199                 continue;
200             }
201             if (tmpLectureScore < 0 || tmpLectureScore > 100)
202             {
203                 std::cout << "输入错误! " << std::endl;
204                 continue;
205             }
206             if (destLectureType == OPTIONAL)
207             {
208                 std::cout << "该课程为任选课, 得分是否计PF? [Y/N]" << std::endl;
209                 int inputLecturePF = _getch();
210                 if (inputLecturePF == 'Y' || inputLecturePF == 'y')
211                 {
212                     tmpLecturePF = true;
213                 }
214             }
215             lectureScore[existLecture - 1] = tmpLectureScore;
216             lecturePF[existLecture - 1] = tmpLectureScore;
217         }
218     }
219     else
220     {
221         if (database.queryLecture(tmpLectureName, false))
222         {

```

```

223         std::cout << "该课程已存在于数据库中。它的类型是";
224         if (database.queryRequired(tmpLectureName, false))
225         {
226             std::cout << "必修课。" << std::endl;
227             destLectureType = REQUIRED;
228             tmpLectureCredit = database.findRequired(tmpLectureName)-
>getCredit();
229         }
230         else if (database.queryLimited(tmpLectureName, false))
231         {
232             std::cout << "限选课。" << std::endl;
233             destLectureType = LIMITED;
234             tmpLectureCredit = database.findLimited(tmpLectureName)-
>getCredit();
235         }
236         else
237         {
238             std::cout << "任选课。" << std::endl;
239             destLectureType = OPTIONAL;
240             tmpLectureCredit = database.findOptional(tmpLectureName)-
>getCredit();
241         }
242     }
243     else
244     {
245         std::cout << "该课程不存在于数据库中。它的类型是什么？" << std::endl;
246         std::cout << "1.必修课" << std::endl
247             << "2.限选课" << std::endl
248             << "3.任选课" << std::endl;
249         tmpLectureType = _getch();
250         if (tmpLectureType != '1' && tmpLectureType != '2' && tmpLectureType
!= '3')
251         {
252             std::cout << "输入错误！" << std::endl;
253             continue;
254         }
255         switch (tmpLectureType)
256         {
257             case '1':
258                 destLectureType = REQUIRED;
259                 break;
260             case '2':
261                 destLectureType = LIMITED;
262                 break;
263             case '3':
264                 destLectureType = OPTIONAL;
265                 break;
266             default:
267                 break;
268         }
269         std::cout << "请输入课程学分： ";
270         std::cin >> tmpLectureCredit;

```

```

271         if (!std::cin)
272         {
273             std::cout << "输入错误! " << std::endl;
274             std::cin.clear();
275             std::cin.ignore();
276             continue;
277         }
278         if (tmpLectureCredit < 0)
279         {
280             std::cout << "输入错误! " << std::endl;
281             continue;
282         }
283         newLecture = true;
284     }
285     std::cout << "请输入课程成绩 (0~100) : ";
286     std::cin >> tmpLectureScore;
287     if (!std::cin)
288     {
289         std::cout << "输入错误! " << std::endl;
290         std::cin.clear();
291         std::cin.ignore();
292         continue;
293     }
294     if (tmpLectureScore < 0 || tmpLectureScore > 100)
295     {
296         std::cout << "输入错误! " << std::endl;
297         continue;
298     }
299     if (destLectureType == OPTIONAL)
300     {
301         std::cout << "该课程为任选课, 得分是否计PF? [Y/N]" << std::endl;
302         int inputLecturePF = _getch();
303         if (inputLecturePF == 'Y' || inputLecturePF == 'y')
304         {
305             tmpLecturePF = true;
306         }
307     }
308     lectureScore.push_back(tmpLectureScore);
309     lectureName.push_back(tmpLectureName);
310     lectureType.push_back(destLectureType);
311     lectureCredit.push_back(tmpLectureCredit);
312     lecturePF.push_back(tmpLecturePF);
313     lectureNum++;
314     if (tmpLecturePF)
315         lecturePFNum++;
316     validLecture = true;
317     tmpLectureGPA = database.calculateGPA(tmpLectureScore);
318     lectureGPA.push_back(tmpLectureGPA);
319     if (!tmpLecturePF)
320         totalScore += tmpLectureScore * tmpLectureCredit;
321     totalCredit += tmpLectureCredit;
322     if (tmpLecturePF)

```



```

323         totalPFCredit += tmpLectureCredit;
324     if (!tmpLecturePF)
325         totalGPA += tmpLectureGPA * tmpLectureCredit;
326     if (totalCredit - totalPFCredit > 0)
327         averageScore = totalScore / (totalCredit - totalPFCredit);
328     if (totalCredit - totalPFCredit > 0)
329         averageGPA = totalGPA / (totalCredit - totalPFCredit);
330     if (newLecture)
331     {
332         switch (tmpLectureType)
333         {
334             case '1':
335                 database.addRequired(tmpLectureName, tmpLectureCredit);
336                 destLectureType = REQUIRED;
337                 break;
338             case '2':
339                 database.addLimited(tmpLectureName, tmpLectureCredit);
340                 destLectureType = LIMITED;
341                 break;
342             case '3':
343                 database.addOptional(tmpLectureName, tmpLectureCredit);
344                 destLectureType = OPTIONAL;
345                 break;
346             default:
347                 break;
348         }
349     }
350     // database.addStudentToLecture(tmpLectureName, destLectureType, *this);
351 }
352
353 std::cout << "是否继续输入课程? [Y/N]" << std::endl;
354 char tmp;
355 tmp = _getch();
356 if (tmp == 'Y' || tmp == 'y')
357 {
358     moreLecture = true;
359 }
360 else
361 {
362     moreLecture = false;
363 }
364 } while (!validLecture || moreLecture);
365 }
366
367 // 设置学号
368 void Student::setStudentNo(int inputStudentNo)
369 {
370     studentNo = inputStudentNo;
371 }
372
373 // 设置姓名
374 void Student::setName(std::string inputName)

```

```
375 {
376     name = inputName;
377 }
378
379 // 设置课程名称
380 void Student::setLectureName(std::vector<std::string> inputLectureName)
381 {
382     lectureName = inputLectureName;
383 }
384
385 // 设置课程类型
386 void Student::setLectureType(std::vector<LectureType> inputLectureType)
387 {
388     lectureType = inputLectureType;
389 }
390
391 // 设置课程学分
392 void Student::setLectureCredit(std::vector<int> inputLectureCredit)
393 {
394     lectureCredit = inputLectureCredit;
395 }
396
397 // 设置课程成绩
398 void Student::setLectureScore(std::vector<int> inputLectureScore)
399 {
400     lectureScore = inputLectureScore;
401 }
402
403 // 设置课程绩点
404 void Student::setLectureGPA(std::vector<double> inputLectureGPA)
405 {
406     lectureGPA = inputLectureGPA;
407 }
408
409 // 设置课程PF
410 void Student::setLecturePF(std::vector<int> inputLecturePF)
411 {
412     lecturePF = inputLecturePF;
413 }
414
415 // 设置课程数
416 void Student::setLectureNum(int inputLectureNum)
417 {
418     lectureNum = inputLectureNum;
419 }
420
421 // 设置PF课程数
422 void Student::setLecturePFNum(int inputLecturePFNum)
423 {
424     lecturePFNum = inputLecturePFNum;
425 }
426
```

```
427 // 设置总学分
428 void Student::setTotalCredit(int inputTotalCredit)
429 {
430     totalCredit = inputTotalCredit;
431 }
432
433 // 设置PF学分
434 void Student::setTotalPFCredit(int inputTotalPFCredit)
435 {
436     totalPFCredit = inputTotalPFCredit;
437 }
438
439 // 设置总成绩
440 void Student::setTotalScore(int inputTotalScore)
441 {
442     totalScore = inputTotalScore;
443 }
444
445 // 设置总绩点
446 void Student::setTotalGPA(double inputTotalGPA)
447 {
448     totalGPA = inputTotalGPA;
449 }
450
451 // 设置平均成绩
452 void Student::setAverageScore(double inputAverageScore)
453 {
454     averageScore = inputAverageScore;
455 }
456
457 // 设置平均绩点
458 void Student::setAverageGPA(double inputAverageGPA)
459 {
460     averageGPA = inputAverageGPA;
461 }
462
463 // 简略打印学生信息
464 void Student::printInfo(int widthStudentNo, int widthName)
465 {
466     std::cout << std::setw(widthStudentNo) << studentNo
467               << std::setw(widthName) << name
468               << std::setw(5) << lectureNum
469               << std::setw(5) << totalCredit
470               << std::setw(5) << std::setprecision(3) << averageScore
471               << std::setw(5) << std::setprecision(3) << averageGPA << std::endl;
472 }
473
474 // 详细打印学生信息
475 void Student::printStudentInfo()
476 {
477     if (isDebugMode())
478         std::cout << "UID: " << uid << std::endl;
```

```

479     std::cout << "学号: " << studentNo << std::endl
480           << "姓名: " << name << std::endl
481           << "课程数: " << lectureNum << std::endl
482           << "总学分: " << totalCredit << std::endl
483           << "平均成绩: " << averageScore << std::endl
484           << "平均绩点: " << averageGPA << std::endl;
485     int widthLectureName = 9, widthLectureType = 5;
486     for (int i = 0; i < lectureNum; i++)
487     {
488         if (lectureName[i].length() >= widthLectureName)
489         {
490             widthLectureName = lectureName[i].length() + 1;
491         }
492     }
493     std::cout << std::setw(widthLectureName) << "课程名称"
494           << std::setw(widthLectureType) << "类型"
495           << std::setw(5) << "学分"
496           << std::setw(5) << "成绩"
497           << std::setw(5) << "绩点" << std::endl;
498     for (int i = 0; i < lectureNum; i++)
499     {
500         if (!lecturePF[i])
501         {
502             std::cout << std::setw(widthLectureName) << lectureName[i]
503                   << std::setw(widthLectureType) << ((lectureType[i] == REQUIRED)
504 ? "必选" : ((lectureType[i] == LIMITED) ? "限选" : "任选"))
505                   << std::setw(5) << lectureCredit[i]
506                   << std::setw(5) << std::setprecision(3) << lectureScore[i]
507                   << std::setw(5) << std::setprecision(3) << lectureGPA[i] <<
508 std::endl;
509         }
510         else
511         {
512             std::cout << std::setw(widthLectureName) << lectureName[i]
513                   << std::setw(widthLectureType) << ((lectureType[i] == REQUIRED)
514 ? "必选" : ((lectureType[i] == LIMITED) ? "限选" : "任选"))
515                   << std::setw(5) << lectureCredit[i]
516                   << std::setw(5) << "N/A"
517                   << std::setw(5) << (lectureGPA[i] ? "P" : "F") << std::endl;
518         }
519     }
520 }
521
522 // 流输入操作符重载函数
523 std::istream &operator>>(std::istream &is, Student &student)
524 {
525     is >> student.studentNo >> student.name >> student.totalCredit >>
526 student.totalScore >> student.totalGPA >> student.averageScore >> student.averageGPA
527 >> student.lectureNum >> student.lecturePFNum;
528     student.lectureName.resize(student.lectureNum);
529     student.lectureType.resize(student.lectureNum);
530     student.lectureCredit.resize(student.lectureNum);

```

```

526     student.lectureScore.resize(student.lectureNum);
527     student.lectureGPA.resize(student.lectureNum);
528     student.lecturePF.resize(student.lectureNum);
529     int tmpLectureType;
530     for (int i = 0; i < student.lectureNum; i++)
531     {
532         is >> student.lectureName[i] >> tmpLectureType >> student.lectureCredit[i] >>
student.lectureScore[i] >> student.lectureGPA[i] >> student.lecturePF[i];
533         student.lectureType[i] = (LectureType)tmpLectureType;
534     }
535     return is;
536 }
537
538 // 流输出操作符重载函数
539 std::ostream &operator<<(std::ostream &os, Student &student)
540 {
541     os << student.studentNo << "\t"
542         << student.name << "\t"
543         << student.totalCredit << "\t"
544         << student.totalScore << "\t"
545         << student.totalGPA << "\t"
546         << student.averageScore << "\t"
547         << student.averageGPA << "\t"
548         << student.lectureNum << "\t"
549         << student.lecturePFNum << std::endl;
550     for (int i = 0; i < student.lectureNum; i++)
551     {
552         os << student.lectureName[i] << "\t"
553             << student.lectureType[i] << "\t"
554             << student.lectureCredit[i] << "\t"
555             << student.lectureScore[i] << "\t"
556             << student.lectureGPA[i] << "\t"
557             << student.lecturePF[i] << std::endl;
558     }
559     return os;
560 }
561

```

## userinterface.cpp

```

1  #include "commonheader.h"
2
3  // 构造函数
4  UserInterface::UserInterface()
5  {
6      database = nullptr;
7  }
8
9  // 析构函数
10 UserInterface::~UserInterface()
11 {

```

```

12     if (database != nullptr)
13     {
14         delete database;
15     }
16 }
17
18 // 主要的交互界面
19 bool UserInterface::run()
20 {
21     system("cls");
22     std::cin.clear();
23     std::cin.sync();
24
25     if (database == nullptr)
26     {
27         database = new Database();
28     }
29     if (currentUser == nullptr)
30         currentUser = login();
31     if (currentUser == nullptr)
32         return false;
33     welcome();
34     int input;
35     input = _getch();
36     if (currentUser->getPermission() == 1)
37     {
38         switch (input)
39         {
40             case '1':
41                 while (searchInfo())
42                     ;
43                 break;
44             case '2':
45                 while (sortInfo())
46                     ;
47                 break;
48             case '3':
49                 while (load())
50                     ;
51                 break;
52             case '4':
53                 while (print())
54                     ;
55                 break;
56             case '5':
57                 while (about())
58                     ;
59                 break;
60             case '0':
61                 quit();
62                 return false;
63         }

```

```
64     }
65     else if (currentUser->getPermission() == 2)
66     {
67         switch (input)
68         {
69             case '1':
70                 while (searchInfo())
71                     ;
72                 break;
73             case '2':
74                 while (sortInfo())
75                     ;
76                 break;
77             case '3':
78                 while (addInfo())
79                     ;
80                 break;
81             case '4':
82                 while (deleteInfo())
83                     ;
84                 break;
85             case '5':
86                 while (modifyInfo())
87                     ;
88                 break;
89             case '6':
90                 while (load())
91                     ;
92                 break;
93             case '7':
94                 while (save())
95                     ;
96                 break;
97             case '8':
98                 while (print())
99                     ;
100                break;
101            case '9':
102                while (about())
103                    ;
104                break;
105            case '0':
106                quit();
107                return false;
108            case '\':
109            case '~':
110                while (debug())
111                    ;
112                break;
113            default:
114                std::cout << "输入错误，请重新输入！" << std::endl;
115                pause();
```

```

116     }
117 }
118
119     return true;
120 }
121
122 // 暂停函数
123 void UserInterface::pause()
124 {
125     std::cin.clear();
126     std::cin.sync();
127     std::cout << "按任意键继续...";
128     _getch();
129 }
130
131 // 登录界面
132 Account *UserInterface::login()
133 {
134     database->loadAccount("savedata_user.dat", "savedata_admin.dat");
135     system("cls");
136     std::cin.clear();
137     std::cin.sync();
138     std::cout << "欢迎使用学生成绩管理系统" << std::endl
139             << "1.登录" << std::endl
140             << "2.注册" << std::endl
141             << "3.退出" << std::endl
142             << "请输入您的选择: " << std::endl;
143     int input;
144     input = _getch();
145     int tmpPermission;
146     std::string tmpUsername;
147     std::string tmpPassword;
148     Account *acc = nullptr;
149     switch (input)
150     {
151     case '1':
152         std::cout << "用户名: " << std::endl;
153         std::cin >> tmpUsername;
154         std::cout << "密码: " << std::endl;
155         std::cin >> tmpPassword;
156         acc = database->login(tmpUsername, tmpPassword);
157         if (acc != nullptr)
158         {
159             std::cout << "登录成功! " << std::endl;
160             database->saveAccount("savedata_user.dat", "savedata_admin.dat");
161             pause();
162             std::cout << std::endl;
163             return acc;
164         }
165     else
166     {
167         std::cout << "登录失败! " << std::endl;

```



```

168         pause();
169         return nullptr;
170     }
171     break;
172 case '2':
173     std::cout << "用户类型: 1.学生 2.教师" << std::endl;
174     tmpPermission = _getch();
175     if (tmpPermission != '1' && tmpPermission != '2')
176     {
177         std::cout << "输入错误, 请重新输入!" << std::endl;
178         pause();
179         return nullptr;
180     }
181     tmpPermission -= 48;
182     if (tmpPermission == 2)
183         std::cout << "用户名: " << std::endl;
184     else
185         std::cout << "学号: " << std::endl;
186     std::cin >> tmpUsername;
187     std::cout << "密码: " << std::endl;
188     std::cin >> tmpPassword;
189     acc = database->registerUser(tmpUsername, tmpPassword, tmpPermission);
190     if (acc != nullptr)
191     {
192         std::cout << "注册成功!" << std::endl;
193         database->saveAccount("savedata_user.dat", "savedata_admin.dat");
194         pause();
195         std::cout << std::endl;
196         return acc;
197     }
198     else
199     {
200         std::cout << "注册失败!" << std::endl;
201         pause();
202         return nullptr;
203     }
204     break;
205 case '3':
206     return nullptr;
207     break;
208 }
209 }
210
211 // 欢迎界面
212 void UserInterface::welcome()
213 {
214     system("cls");
215     std::cout << "< - 学生成绩管理系统 - >" << std::endl
216         << std::endl;
217     if (currentUser->getPermission() == 1)
218     {
219         if (Info::isDebugMode())

```

```

220         std::cout << "~.调试模式" << std::endl;
221     std::cout << "1.查询信息" << std::endl
222         << "2.全部信息" << std::endl
223         << "3.读取信息" << std::endl
224         << "4.打印信息" << std::endl
225         << "5.关于程序" << std::endl
226         << "0.退出程序" << std::endl
227         << "请输入您的选择: " << std::endl;
228 }
229 else if (currentUser->getPermission() == 2)
230 {
231     if (Info::isDebugMode())
232         std::cout << "~.调试模式" << std::endl;
233     std::cout << "1.查询信息" << std::endl
234         << "2.全部信息" << std::endl
235         << "3.添加信息" << std::endl
236         << "4.删除信息" << std::endl
237         << "5.修改信息" << std::endl
238         << "6.读取信息" << std::endl
239         << "7.保存信息" << std::endl
240         << "8.打印信息" << std::endl
241         << "9.关于程序" << std::endl
242         << "0.退出程序" << std::endl
243         << "请输入您的选择: " << std::endl;
244 }
245 else
246 {
247     std::cout << "未知用户类型!" << std::endl;
248 }
249 }
250
251 // 1.查询信息
252 bool UserInterface::searchInfo()
253 {
254     system("cls");
255
256     std::cout << "< - 查询信息 - >" << std::endl
257         << std::endl;
258     std::cout << "1.查询学生信息" << std::endl
259         << "2.查询课程信息" << std::endl
260         << "3.返回上一级" << std::endl
261         << "请输入您的选择: " << std::endl;
262
263     int input;
264     input = _getch();
265
266     switch (input)
267     {
268     case '1':
269         while (searchStudent())
270             ;
271         break;

```

```

272     case '2':
273         while (searchLecture())
274             ;
275         break;
276     case '3':
277         std::cout << "返回上一级。" << std::endl;
278         return false;
279     default:
280         std::cout << "输入错误, 请重新输入!" << std::endl;
281         break;
282     }
283     pause();
284     return true;
285 }
286
287 // 查询学生信息
288 bool UserInterface::searchStudent()
289 {
290     system("cls");
291
292     if (currentUser->getPermission() == 1)
293     {
294         std::cout << "< - 查询学生信息 - >" << std::endl
295             << std::endl;
296         int num = database->queryStudent(atoi(currentUser->getName().c_str()));
297         if (num == 0)
298         {
299             std::cout << "未找到该学生!" << std::endl;
300         }
301         else
302         {
303             std::cout << "共找到" << num << "个学生。" << std::endl;
304         }
305         return false;
306     }
307     else
308     {
309         std::cout << "< - 查询学生信息 - >" << std::endl
310             << std::endl;
311         std::cout << "1.按学号查询" << std::endl
312             << "2.按姓名查询" << std::endl
313             << "3.返回上一级" << std::endl
314             << "请输入您的选择:" << std::endl;
315
316         int input;
317         int num;
318         int stuNo;
319         std::string stuName;
320         input = _getch();
321
322         switch (input)
323         {

```

```

324     case '1':
325         std::cout << "请输入学号: ";
326         std::cin >> stuNo;
327         if (!std::cin)
328         {
329             std::cout << "输入错误, 请重新输入!" << std::endl;
330             pause();
331             return true;
332         }
333         num = database->queryStudent(stuNo);
334         if (num == 0)
335         {
336             std::cout << "未找到该学生!" << std::endl;
337         }
338         else
339         {
340             std::cout << "共找到" << num << "个学生." << std::endl;
341         }
342         break;
343     case '2':
344         std::cout << "请输入姓名: ";
345         std::cin >> stuName;
346         if (!std::cin)
347         {
348             std::cout << "输入错误, 请重新输入!" << std::endl;
349             pause();
350             return true;
351         }
352         num = database->queryStudent(stuName);
353         if (num == 0)
354         {
355             std::cout << "未找到该学生!" << std::endl;
356         }
357         else
358         {
359             std::cout << "共找到" << num << "个学生." << std::endl;
360         }
361         break;
362     case '3':
363         std::cout << "返回上一级." << std::endl;
364         return false;
365     default:
366         std::cout << "输入错误, 请重新输入!" << std::endl;
367         break;
368     }
369     pause();
370     return true;
371 }
372 }
373
374 // 查询课程信息
375 bool UserInterface::searchLecture()

```

```

376 {
377     system("cls");
378
379     std::cout << "< - 查询课程信息 - >" << std::endl
380         << std::endl;
381     std::cout << "1.按课程名查询" << std::endl
382         << "2.返回上一级" << std::endl
383         << "请输入您的选择: " << std::endl;
384
385     int input = 0;
386     int num = 0;
387     std::string lecName;
388     input = _getch();
389
390     if (!std::cin)
391     {
392         std::cout << "输入错误, 请重新输入! " << std::endl;
393         pause();
394         return true;
395     }
396
397     switch (input)
398     {
399     case '1':
400         std::cout << "请输入课程名: ";
401         std::cin >> lecName;
402         if (!std::cin)
403         {
404             std::cout << "输入错误, 请重新输入! " << std::endl;
405             pause();
406             return true;
407         }
408         num = database->queryLecture(lecName);
409         if (num == 0)
410         {
411             std::cout << "未找到该课程! " << std::endl;
412         }
413         else
414         {
415             std::cout << "共找到" << num << "个课程。" << std::endl;
416         }
417         break;
418     case '2':
419         std::cout << "返回上一级。" << std::endl;
420         return false;
421     default:
422         std::cout << "输入错误, 请重新输入! " << std::endl;
423         pause();
424         return true;
425     }
426     pause();
427     return true;

```

```

428 }
429
430 // 全部信息
431 bool UserInterface::sortInfo()
432 {
433     system("cls");
434
435     if (currentUser->getPermission() == 1)
436     {
437         while (sortLecture())
438             ;
439         return false;
440     }
441     else
442     {
443         std::cout << "< - 全部信息 - >" << std::endl
444             << std::endl;
445         std::cout << "1.显示学生信息" << std::endl
446             << "2.显示课程信息" << std::endl
447             << "3.返回上一级" << std::endl
448             << "请输入您的选择: " << std::endl;
449
450         int input;
451         input = _getch();
452
453         switch (input)
454         {
455             case '1':
456                 while (sortStudent())
457                     ;
458                 break;
459             case '2':
460                 while (sortLecture())
461                     ;
462                 break;
463             case '3':
464                 std::cout << "返回上一级。" << std::endl;
465                 return false;
466             default:
467                 std::cout << "输入错误, 请重新输入! " << std::endl;
468                 break;
469         }
470     }
471     pause();
472     return true;
473 }
474
475 // 排序学生信息
476 bool UserInterface::sortStudent()
477 {
478     system("cls");
479

```

```

480     int input = 0;
481     int direction = 0;
482
483     std::cout << "< - 全部学生信息 - >" << std::endl
484         << std::endl;
485     std::cout << "1.默认排序" << std::endl
486         << "2.按学号排序" << std::endl
487         << "3.按姓名排序" << std::endl
488         << "4.按总成绩排序" << std::endl
489         << "5.按总学分排序" << std::endl
490         << "6.按总绩点排序" << std::endl
491         << "7.按平均成绩排序" << std::endl
492         << "8.按GPA排序" << std::endl
493         << "9.返回上一级" << std::endl
494         << "请输入您的选择: " << std::endl;
495     input = _getch();
496     switch (input)
497     {
498     case '1':
499     case '2':
500     case '3':
501     case '4':
502     case '5':
503     case '6':
504     case '7':
505     case '8':
506         break;
507     case '9':
508         std::cout << "返回上一级。" << std::endl;
509         return false;
510     default:
511         std::cout << "输入错误, 请重新输入! " << std::endl;
512         pause();
513         return true;
514     }
515
516     if (input != '1')
517     {
518         std::cout << "排序方向选择" << std::endl
519             << "1.升序" << std::endl
520             << "2.降序" << std::endl
521             << "请输入您的选择: " << std::endl;
522         direction = _getch();
523         if (direction != '1' && direction != '2')
524         {
525             std::cout << "输入错误, 请重新输入! " << std::endl;
526             pause();
527             return true;
528         }
529     }
530
531     database->sortStudent((direction == '1') ? -1 : 1, input - 50);

```

```

532
533     pause();
534     return true;
535 }
536
537 // 排序课程信息
538 bool UserInterface::sortLecture()
539 {
540     system("cls");
541
542     int input = 0;
543     int direction = 0;
544     int lecType = 0;
545
546     std::cout << "< - 全部课程信息 - >" << std::endl
547         << std::endl;
548     std::cout << "1.默认排序" << std::endl
549         << "2.按课程名排序" << std::endl
550         << "3.按学分排序" << std::endl
551         << "4.按学生数排序" << std::endl
552         << "5.按平均分排序" << std::endl
553         << "6.按平均绩点排序" << std::endl
554         << "7.返回上一级" << std::endl
555         << "请输入您的选择: " << std::endl;
556     input = _getch();
557     switch (input)
558     {
559     case '1':
560     case '2':
561     case '3':
562     case '4':
563     case '5':
564     case '6':
565         break;
566     case '7':
567         std::cout << "返回上一级。" << std::endl;
568         return false;
569     default:
570         std::cout << "输入错误, 请重新输入! " << std::endl;
571         pause();
572         return true;
573     }
574
575     std::cout << "课程类型选择" << std::endl
576         << "1.全部课" << std::endl
577         << "2.必修课" << std::endl
578         << "3.限选课" << std::endl
579         << "4.任选课" << std::endl
580         << "请输入您的选择: " << std::endl;
581     lecType = _getch();
582     if (lecType != '1' && lecType != '2' && lecType != '3' && lecType != '4')
583     {

```



```

584         std::cout << "输入错误, 请重新输入! " << std::endl;
585         pause();
586         return true;
587     }
588
589     if (input != '1')
590     {
591         std::cout << "排序方向选择" << std::endl
592             << "1.升序" << std::endl
593             << "2.降序" << std::endl
594             << "请输入您的选择: " << std::endl;
595         direction = _getch();
596         if (direction != '1' && direction != '2')
597         {
598             std::cout << "输入错误, 请重新输入! " << std::endl;
599             pause();
600             return true;
601         }
602     }
603
604     database->sortLecture((LectureType)(lecType - 49), (direction == '1') ? -1 : 1,
        input - 50);
605
606     pause();
607     return true;
608 }
609
610 // 添加信息
611 bool UserInterface::addInfo()
612 {
613     system("cls");
614
615     std::cout << "< - 添加信息 - >" << std::endl
616         << std::endl;
617     std::cout << "1.添加学生信息" << std::endl
618         << "2.返回上一级" << std::endl
619         << "请输入您的选择: " << std::endl;
620
621     int input;
622     input = _getch();
623
624     switch (input)
625     {
626     case '1':
627         while (addStudent())
628             ;
629         break;
630     case '2':
631         std::cout << "返回上一级。" << std::endl;
632         return false;
633     default:
634         std::cout << "输入错误, 请重新输入! " << std::endl;

```

```

635         break;
636     }
637     pause();
638     return true;
639 }
640
641 // 添加学生信息
642 bool UserInterface::addStudent()
643 {
644     system("cls");
645
646     std::cout << "< - 添加学生信息 - >" << std::endl
647         << std::endl;
648     database->addStudent();
649     database->updateLecture();
650
651     std::cout << "是否继续添加? [Y/N]" << std::endl;
652     char input;
653     input = _getch();
654     if (input == 'Y' || input == 'y')
655         return true;
656     else
657         return false;
658 }
659
660 // 添加课程信息
661 bool UserInterface::addLecture()
662 {
663     return true;
664 }
665
666 // 删除信息
667 bool UserInterface::deleteInfo()
668 {
669     system("cls");
670
671     std::cout << "< - 删除学生信息 - >" << std::endl
672         << std::endl;
673     std::cout << "1.删除学生信息" << std::endl
674         << "2.删除课程信息" << std::endl
675         << "3.返回上一级" << std::endl
676         << "请输入您的选择: " << std::endl;
677
678     int input;
679     input = _getch();
680
681     switch (input)
682     {
683     case '1':
684         while (deleteStudent())
685             ;
686         break;

```

```

687     case '2':
688         while (deleteLecture())
689             ;
690         break;
691     case '3':
692         std::cout << "返回上一级。" << std::endl;
693         return false;
694     default:
695         std::cout << "输入错误, 请重新输入!" << std::endl;
696         break;
697     }
698     pause();
699     return true;
700 }
701
702 // 删除学生信息
703 bool UserInterface::deleteStudent()
704 {
705     system("cls");
706
707     std::cout << "< - 删除学生信息 - >" << std::endl
708         << std::endl;
709     std::cout << "1.按学号删除" << std::endl
710         << "2.按姓名删除" << std::endl
711         << "3.返回上一级" << std::endl
712         << "请输入您的选择: " << std::endl;
713
714     int input;
715     int num;
716     int stuNo;
717     std::string stuName;
718
719     input = _getch();
720     if (!std::cin)
721     {
722         std::cout << "输入错误, 请重新输入!" << std::endl;
723         pause();
724         return true;
725     }
726     switch (input)
727     {
728     case '1':
729         std::cout << "请输入学号: ";
730         std::cin >> stuNo;
731         if (!std::cin)
732         {
733             std::cout << "输入错误, 请重新输入!" << std::endl;
734             pause();
735             return true;
736         }
737         num = database->queryStudent(stuNo, false);
738         if (num == 0)

```

```

739     {
740         std::cout << "未找到该学生！" << std::endl;
741     }
742     else
743     {
744         database->deleteStudent(stuNo);
745         std::cout << "已删除" << num << "个学生。" << std::endl;
746         database->updateLecture();
747     }
748     break;
749 case '2':
750     std::cout << "请输入姓名：";
751     std::cin >> stuName;
752     if (!std::cin)
753     {
754         std::cout << "输入错误，请重新输入！" << std::endl;
755         pause();
756         return true;
757     }
758     num = database->queryStudent(stuName, false);
759     if (num == 0)
760     {
761         std::cout << "未找到该学生！" << std::endl;
762     }
763     else
764     {
765         database->deleteStudent(stuName);
766         std::cout << "已删除" << num << "个学生。" << std::endl;
767         database->updateLecture();
768     }
769     break;
770 case '3':
771     std::cout << "返回上一级。" << std::endl;
772     return false;
773 default:
774     std::cout << "输入错误，请重新输入！" << std::endl;
775     break;
776 }
777
778 std::cout << "是否继续删除？[Y/N]" << std::endl;
779 char ctn;
780 ctn = _getch();
781 if (ctn == 'Y' || ctn == 'y')
782     return true;
783 else
784     return false;
785 }
786
787 // 删除课程信息
788 bool UserInterface::deleteLecture()
789 {
790     system("cls");

```

```

791
792     std::cout << "< - 删除课程信息 - >" << std::endl
793         << std::endl;
794     std::cout << "1.按名称删除" << std::endl
795         << "2.返回上一级" << std::endl
796         << "请输入您的选择: " << std::endl;
797
798     int input;
799     int num;
800     std::string lecName;
801
802     input = _getch();
803     switch (input)
804     {
805     case '1':
806         std::cout << "请输入课程名: ";
807         std::cin >> lecName;
808         if (!std::cin)
809         {
810             std::cout << "输入错误, 请重新输入! " << std::endl;
811             pause();
812             return true;
813         }
814         num = database->queryLecture(lecName, false);
815         if (num == 0)
816         {
817             std::cout << "未找到该课程! " << std::endl;
818         }
819         else
820         {
821             if (database->queryRequired(lecName, false))
822             {
823                 database->deleteRequired(lecName);
824                 std::cout << "已删除" << num << "个必修课。" << std::endl;
825                 database->updateStudent();
826             }
827             else if (database->queryLimited(lecName, false))
828             {
829                 database->deleteLimited(lecName);
830                 std::cout << "已删除" << num << "个限选课。" << std::endl;
831                 database->updateStudent();
832             }
833             else
834             {
835                 database->deleteOptional(lecName);
836                 std::cout << "已删除" << num << "个任选课。" << std::endl;
837                 database->updateStudent();
838             }
839         }
840         break;
841     case '2':
842         std::cout << "返回上一级。" << std::endl;

```

```

843         return false;
844     default:
845         std::cout << "输入错误, 请重新输入! " << std::endl;
846         break;
847     }
848
849     std::cout << "是否继续删除? [Y/N]" << std::endl;
850     char ctn;
851     ctn = _getch();
852     if (ctn == 'Y' || ctn == 'y')
853         return true;
854     else
855         return false;
856 }
857
858 // 修改信息
859 bool UserInterface::modifyInfo()
860 {
861     system("cls");
862
863     std::cout << "< - 修改信息 - >" << std::endl
864         << std::endl;
865     std::cout << "1.修改学生信息" << std::endl
866         << "2.修改课程信息" << std::endl
867         << "3.返回上一级" << std::endl
868         << "请输入您的选择: " << std::endl;
869
870     int input;
871     input = _getch();
872     switch (input)
873     {
874     case '1':
875         while (modifyStudent())
876             ;
877         break;
878     case '2':
879         while (modifyLecture())
880             ;
881         break;
882     case '3':
883         std::cout << "返回上一级。" << std::endl;
884         return false;
885     default:
886         std::cout << "输入错误, 请重新输入! " << std::endl;
887         break;
888     }
889
890     std::cout << "是否继续修改? [Y/N]" << std::endl;
891     char ctn;
892     ctn = _getch();
893     if (ctn == 'Y' || ctn == 'y')
894         return true;

```

```

895     else
896         return false;
897 }
898
899 // 修改学生信息
900 bool UserInterface::modifyStudent()
901 {
902     system("cls");
903
904     std::cout << "< - 修改学生信息 - >" << std::endl
905         << std::endl;
906     std::cout << "1.按学号修改" << std::endl
907         << "2.按姓名修改" << std::endl
908         << "3.返回上一级" << std::endl
909         << "请输入您的选择: " << std::endl;
910
911     int input;
912     int num;
913     int stuNo;
914     std::string stuName;
915
916     input = _getch();
917     switch (input)
918     {
919     case '1':
920         std::cout << "请输入学号: ";
921         std::cin >> stuNo;
922         if (!std::cin)
923         {
924             std::cout << "输入错误, 请重新输入!" << std::endl;
925             pause();
926             return true;
927         }
928         num = database->queryStudent(stuNo, false);
929         if (num == 0)
930         {
931             std::cout << "未找到该学生!" << std::endl;
932         }
933         else
934         {
935             database->modifyStudent(stuNo);
936             std::cout << "已修改" << num << "个学生." << std::endl;
937             database->updateLecture();
938         }
939         break;
940     case '2':
941         std::cout << "请输入姓名: ";
942         std::cin >> stuName;
943         if (!std::cin)
944         {
945             std::cout << "输入错误, 请重新输入!" << std::endl;
946             pause();

```

```

947         return true;
948     }
949     num = database->queryStudent(stuName, false);
950     if (num == 0)
951     {
952         std::cout << "未找到该学生!" << std::endl;
953     }
954     else
955     {
956         database->modifyStudent(stuName);
957         std::cout << "已修改" << num << "个学生。" << std::endl;
958         database->updateLecture();
959     }
960     break;
961 case '3':
962     std::cout << "返回上一级。" << std::endl;
963     return false;
964 default:
965     std::cout << "输入错误, 请重新输入!" << std::endl;
966     break;
967 }
968
969 std::cout << "是否继续修改? [Y/N]" << std::endl;
970 char ctn;
971 ctn = _getch();
972 if (ctn == 'Y' || ctn == 'y')
973     return true;
974 else
975     return false;
976 }
977
978 // 修改课程信息
979 bool UserInterface::modifyLecture()
980 {
981     system("cls");
982
983     std::cout << "< - 修改课程信息 - >" << std::endl
984         << std::endl;
985     std::cout << "1.按名称修改" << std::endl
986         << "2.返回上一级" << std::endl
987         << "请输入您的选择: " << std::endl;
988
989     int input;
990     std::string lecName;
991
992     input = _getch();
993     switch (input)
994     {
995     case '1':
996         std::cout << "请输入课程名称: ";
997         std::cin >> lecName;
998         if (!std::cin)

```



```

999         {
1000             std::cout << "输入错误, 请重新输入! " << std::endl;
1001             pause();
1002             return true;
1003         }
1004         database->modifyLecture(lecName);
1005         database->updateStudent();
1006         break;
1007     case '2':
1008         std::cout << "返回上一级。" << std::endl;
1009         return false;
1010     default:
1011         std::cout << "输入错误, 请重新输入! " << std::endl;
1012         break;
1013     }
1014
1015     std::cout << "是否继续修改? [Y/N]" << std::endl;
1016     char ctn;
1017     ctn = _getch();
1018     if (ctn == 'Y' || ctn == 'y')
1019         return true;
1020     else
1021         return false;
1022 }
1023
1024 // 读取信息
1025 bool UserInterface::load()
1026 {
1027     system("cls");
1028
1029     std::cout << "< - 读取保存信息 - >" << std::endl
1030         << std::endl;
1031     std::cout << "该操作将会覆盖当前数据, 是否继续? [Y/N]" << std::endl;
1032     int input;
1033     input = _getch();
1034     if (input == 'Y' || input == 'y')
1035     {
1036         try
1037         {
1038             database->load();
1039         }
1040         catch (FileNotFoundException e)
1041         {
1042             if (e.mode == "open")
1043             {
1044                 std::cout << "以" << e.type << "方式打开文件" << e.filename << "失败! "
1045                     << std::endl;
1046             }
1047             else
1048             {
1049                 if (e.type == "read")
1050                 {

```

```

1050         std::cout << "读取文件" << e.filename << "失败！" << std::endl;
1051     }
1052     else
1053     {
1054         std::cout << "写入文件" << e.filename << "失败！" << std::endl;
1055     }
1056 }
1057 }
1058     std::cout << "数据已读取。" << std::endl;
1059 }
1060 pause();
1061 return false;
1062 }
1063
1064 // 保存信息
1065 bool UserInterface::save()
1066 {
1067     system("cls");
1068
1069     std::cout << "< - 保存信息 - >" << std::endl
1070         << std::endl;
1071
1072     try
1073     {
1074         database->save();
1075     }
1076     catch (FileNotFoundException e)
1077     {
1078         if (e.mode == "open")
1079         {
1080             std::cout << "以" << e.type << "方式打开文件" << e.filename << "失败！" <<
std::endl;
1081         }
1082         else
1083         {
1084             if (e.type == "read")
1085             {
1086                 std::cout << "读取文件" << e.filename << "失败！" << std::endl;
1087             }
1088             else
1089             {
1090                 std::cout << "写入文件" << e.filename << "失败！" << std::endl;
1091             }
1092         }
1093     }
1094     std::cout << "数据已保存。" << std::endl;
1095     pause();
1096     return false;
1097 }
1098
1099 // 打印信息
1100 bool UserInterface::print()

```

```

1101 {
1102     system("cls");
1103
1104     if (currentUser->getPermission() == 1)
1105     {
1106         while (printStudent())
1107             ;
1108         return false;
1109     }
1110     else
1111     {
1112         std::cout << "< - 打印信息 - >" << std::endl
1113             << std::endl;
1114         std::cout << "1.打印学生信息" << std::endl
1115             << "2.打印课程信息" << std::endl
1116             << "3.返回上一级" << std::endl
1117             << "请输入您的选择: " << std::endl;
1118
1119         int input;
1120         input = _getch();
1121         switch (input)
1122         {
1123             case '1':
1124                 while (printStudent())
1125                     ;
1126                 break;
1127             case '2':
1128                 while (printLecture())
1129                     ;
1130                 break;
1131             case '3':
1132                 std::cout << "返回上一级。" << std::endl;
1133                 return false;
1134             default:
1135                 std::cout << "输入错误, 请重新输入! " << std::endl;
1136                 pause();
1137                 break;
1138         }
1139     }
1140     pause();
1141     return true;
1142 }
1143
1144 // 打印学生信息
1145 bool UserInterface::printStudent()
1146 {
1147     system("cls");
1148
1149     if (currentUser->getPermission() == 1)
1150     {
1151         std::cout << "< - 打印学生信息 - >" << std::endl
1152             << std::endl;

```

```

1153     database->printStudent(atoi(currentUser->getName().c_str()));
1154     pause();
1155     return false;
1156 }
1157 else
1158 {
1159     std::cout << "< - 打印学生信息 - >" << std::endl
1160         << std::endl;
1161     std::cout << "1.打印全部学生信息" << std::endl
1162         << "2.按学号打印学生信息" << std::endl
1163         << "3.按姓名打印学生信息" << std::endl
1164         << "4.返回上一级" << std::endl
1165         << "请输入您的选择: " << std::endl;
1166
1167     int input;
1168     int stuId;
1169     std::string stuName;
1170
1171     input = _getch();
1172     switch (input)
1173     {
1174     case '1':
1175         database->printStudent();
1176         pause();
1177         break;
1178     case '2':
1179         std::cout << "请输入学号: ";
1180         std::cin >> stuId;
1181         if (!std::cin)
1182         {
1183             std::cout << "输入错误, 请重新输入!" << std::endl;
1184             pause();
1185             return true;
1186         }
1187         database->printStudent(stuId);
1188         pause();
1189         break;
1190     case '3':
1191         std::cout << "请输入姓名: ";
1192         std::cin >> stuName;
1193         if (!std::cin)
1194         {
1195             std::cout << "输入错误, 请重新输入!" << std::endl;
1196             pause();
1197             return true;
1198         }
1199         database->printStudent(stuName);
1200         pause();
1201         break;
1202     case '4':
1203         std::cout << "返回上一级。" << std::endl;
1204         return false;

```

```

1205         default:
1206             std::cout << "输入错误, 请重新输入! " << std::endl;
1207             pause();
1208             break;
1209     }
1210 }
1211 return true;
1212 }
1213
1214 // 打印课程信息
1215 bool UserInterface::printLecture()
1216 {
1217     system("cls");
1218
1219     std::cout << "< - 打印课程信息 - >" << std::endl
1220         << std::endl;
1221     std::cout << "1.打印全部课程信息" << std::endl
1222         << "2.按类型打印课程信息" << std::endl
1223         << "3.按名称打印课程信息" << std::endl
1224         << "4.返回上一级" << std::endl
1225         << "请输入您的选择: " << std::endl;
1226
1227     int input;
1228     int lecType;
1229     std::string lecName;
1230
1231     input = _getch();
1232     switch (input)
1233     {
1234     case '1':
1235         database->printLecture();
1236         pause();
1237         break;
1238     case '2':
1239         std::cout << "请选择课程类型: " << std::endl
1240             << "1.必修课" << std::endl
1241             << "2.限选课" << std::endl
1242             << "3.任选课" << std::endl;
1243         lecType = _getch();
1244         if (lecType < '1' || lecType > '3')
1245         {
1246             std::cout << "输入错误, 请重新输入! " << std::endl;
1247             pause();
1248             return true;
1249         }
1250         database->printLecture((LectureType)(lecType - 48));
1251         pause();
1252         break;
1253     case '3':
1254         std::cout << "请输入课程名: ";
1255         std::cin >> lecName;
1256         if (!std::cin)

```

```

1257     {
1258         std::cout << "输入错误, 请重新输入! " << std::endl;
1259         pause();
1260         return true;
1261     }
1262     database->printLecture(lecName);
1263     pause();
1264     break;
1265 case '4':
1266     std::cout << "返回上一级。" << std::endl;
1267     return false;
1268 default:
1269     std::cout << "输入错误, 请重新输入! " << std::endl;
1270     break;
1271 }
1272 return true;
1273 }
1274
1275 // 关于程序
1276 bool UserInterface::about()
1277 {
1278     system("cls");
1279
1280     std::cout << "< - 关于程序 - >" << std::endl
1281         << std::endl;
1282     std::cout << "学生信息管理系统" << std::endl
1283         << "Copyright (c) Xuc Pan 2023" << std::endl
1284         << "https://github.com/Panxuc" << std::endl;
1285     std::cout << "统计信息: " << std::endl
1286         << "学生" << database->getStudentListSize() << "人。" << std::endl
1287         << "必修课" << database->getRequiredListSize() << "门。" << std::endl
1288         << "限选课" << database->getLimitedListSize() << "门。" << std::endl
1289         << "任选课" << database->getOptionalListSize() << "门。" << std::endl;
1290     pause();
1291     return false;
1292 }
1293
1294 // 退出程序
1295 void UserInterface::quit()
1296 {
1297     if (currentUser->getPermission() == 2)
1298     {
1299         std::cout << "是否保存数据? [Y/N]" << std::endl;
1300         char input;
1301         input = _getch();
1302         if (input == 'Y' || input == 'y')
1303         {
1304             try
1305             {
1306                 database->save();
1307             }
1308             catch (FileNotFoundException e)

```

```

1309         {
1310             if (e.mode == "open")
1311             {
1312                 std::cout << "以" << e.type << "方式打开文件" << e.filename << "失
失败! " << std::endl;
1313             }
1314             else
1315             {
1316                 if (e.type == "read")
1317                 {
1318                     std::cout << "读取文件" << e.filename << "失败! " <<
std::endl;
1319                 }
1320                 else
1321                 {
1322                     std::cout << "写入文件" << e.filename << "失败! " <<
std::endl;
1323                 }
1324             }
1325         }
1326         std::cout << "数据已保存。" << std::endl;
1327     }
1328     else
1329     {
1330         std::cout << "数据未保存。" << std::endl;
1331     }
1332 }
1333 std::cout << "感谢使用本学生成绩管理系统。" << std::endl;
1334 }
1335
1336 //~. 调试模式
1337 bool UserInterface::debug()
1338 {
1339     system("cls");
1340
1341     std::cout << "< - 调试模式 - >" << std::endl
<< std::endl;
1342     if (!Info::isDebugMode())
1343     {
1344         std::cout << "调试模式会输出更多信息，也可以自由修改所有数值，是否继续? [Y/N]" <<
std::endl;
1345         int input;
1346         input = _getch();
1347         if (input == 'Y' || input == 'y')
1348         {
1349             std::cout << "请输入密码: " << std::endl;
1350             std::string password;
1351             std::cin >> password;
1352             if (password == "114514")
1353             {
1354                 std::cout << "调试模式已开启。" << std::endl;
1355                 Info::setDebug(true);

```

```
1357         pause();
1358         return false;
1359     }
1360     else
1361     {
1362         std::cout << "密码错误，调试模式未开启。" << std::endl;
1363         pause();
1364         return false;
1365     }
1366 }
1367 else
1368 {
1369     std::cout << "回到主菜单。" << std::endl;
1370     pause();
1371     return false;
1372 }
1373 }
1374 else
1375 {
1376     std::cout << "调试模式已开启，是否关闭? [Y/N]" << std::endl;
1377     int input;
1378     input = _getch();
1379     if (input == 'Y' || input == 'y')
1380     {
1381         std::cout << "请输入密码: " << std::endl;
1382         std::string password;
1383         std::cin >> password;
1384         if (password == "114514")
1385         {
1386             std::cout << "调试模式已关闭。" << std::endl;
1387             Info::setDebug(false);
1388             pause();
1389             return false;
1390         }
1391         else
1392         {
1393             std::cout << "密码错误，调试模式未关闭。" << std::endl;
1394             pause();
1395             return false;
1396         }
1397     }
1398     else
1399     {
1400         std::cout << "回到主菜单。" << std::endl;
1401         pause();
1402         return false;
1403     }
1404 }
1405 }
1406 }
```