课 程 设 计 报 告

**课程名称 计算机程序设计基础（2）**

**班 级 无36**

**学 号 2013011189**

**姓 名 郭一隆**

**指导助教**

**2014年7月21日**

1. 系统需求分析

学生成绩管理系统作为管理系统，应具有用户及登录功能。用户种类包括管理员、教师、学生、助教等。不同用户具有的权限不同，登入学生成绩管理系统后进行不同操作。详细功能如下表1.1：

|  |  |  |
| --- | --- | --- |
| **用户种类** | **一级功能** | **具体功能** |
| 共有 | 个人中心 | 修改密码 |
| 退出登录 |
| 退出系统 | 退出系统 |
| 管理员 | 用户管理 | 新用户注册 |
| 删除用户 |
| 教师 | 课程管理 | 查看各课程 |
| 增开课程 |
| 删除课程 |
| 查看班级 | 查看班级 |
| 学生 | 课程管理 | 查看已选所有课程 |
| 增加课程 |
| 退课 |
| 助教 | // 同时拥有教师和学生的课程管理功能 | |

表1.1

功能具体需求：

* 1. 登录界面：用户从键盘输入用户名和密码，检验是否正确，返回相应信息。登录成功则进入用户菜单界面。
  2. 修改密码：输入原密码，新密码重复输入两次，检验原密码是否符合，新密码是否符合一定规则（如6~15位）。
  3. 退出登录：退回至用户登录界面。
  4. 退出系统：关闭程序。
  5. 新用户注册：管理员选定新用户的身份，新ID号由系统计算得出（大于当前所有ID号），新密码为默认密码。
  6. 删除用户：输入ID，检验ID是否存在，存在则删除该用户数据。
  7. 教师查看课程：显示课程基本信息（课程号、名称、学分等）、授课教师、学生清单。同时具有录入成绩的功能。
  8. 教师增加课程：输入新课程号，如果课程已存在则直接添加至该教师所开课程，否则进一步输入新开课程信息。
  9. 教师删除课程：从已开课程中删除课程，并删除相关学生的部分信息。
  10. 查看班级：班主任教师可查看班级内学生的成绩概况。
  11. 学生查看课程：查看已选课程基本信息以及自己该课成绩。
  12. 学生增加课程：从教师所开课程中选课。
  13. 学生退课：退课。

1. 总体设计

学生成绩管理系统主要模块：登录模块、菜单模块、数据的添加与删除模块、成绩管理模块。

登录模块：加载登录界面前将所有用户数据从文件读入，以全局数组的形式保存，获取用户输入的信息，与全局数组比对验证。

菜单模块：通过调用<windows.h>中的函数改变控制台字体颜色并实现方向键控制菜单选项。

数据的添加和删除模块：可以增减用户、增减课程。

成绩管理模块：教师、助教以课程为单位录入成绩，学生可以查看成绩以及在相应课程内的排名，班主任查看班级学生成绩情况，以GPA排名。

菜单

学生成绩管理系统

成绩排名

查看成绩

增减课程

录入成绩

增减用户

登录

图2.1 系统功能模块图

1. 详细设计

学生成绩管理系统中9个类/结构体的层次图如下图：

派 生

Score

Course

管理菜单链表

MenuNode

登录验证

Token

TeachingAssistant

Student

Teacher

Administrator

User

图3.1 类/结构体的层次图

* 1. 登录：从键盘输入获取登录令牌（Token）并通过Token::auth()方法验证，返回User\*基类指针。输入密码时键入的字符以‘\*’回显，支持退格键及左右方向键，遇到不合法的输入则输出‘\a’发出alert。
  2. 菜单界面：本系统亮点之一，显示菜单函数MenuNode::show(int k)每行打印一条菜单选项，其中第k条高亮打印。另一方面，利用getch()非缓冲地读入用户的方向键指令，通过调用自定义的清屏函数（系统清屏+打印系统标题）ClearScreen()，并更改高亮显示的选项，以达到仿图形化菜单的视觉效果及用户体验。
  3. 用户管理：键盘读入新用户相应信息，并以std::ios::app方式存至文件，再根据文件更新全局数组。删除则从全局数组中删除相应用户，再根据全局数组更新文件。
  4. 修改密码：在原密码验证通过的前提下，将新密码存入全局数组，紧接着更新文件。
  5. 教师开课：从键盘录入新课程信息，更新教师所开课程ID，并更新文件。
  6. 学生选课：键盘输入课程ID以添加课程，更新学生所选课程ID，并更新文件。
  7. 录入成绩：依次在屏幕显示学生信息，并请求输入该学生成绩，直至录入该课程所有学生的成绩。
  8. 修改成绩：重新录入成绩即可，新成绩会覆盖旧成绩。
  9. 查看课程：教师查看则展示基本信息及所有学生该课程成绩信息；学生查看则展示基本信息和自身成绩和排名信息。
  10. 查看班级：全班学生按GPA高低依次显示。

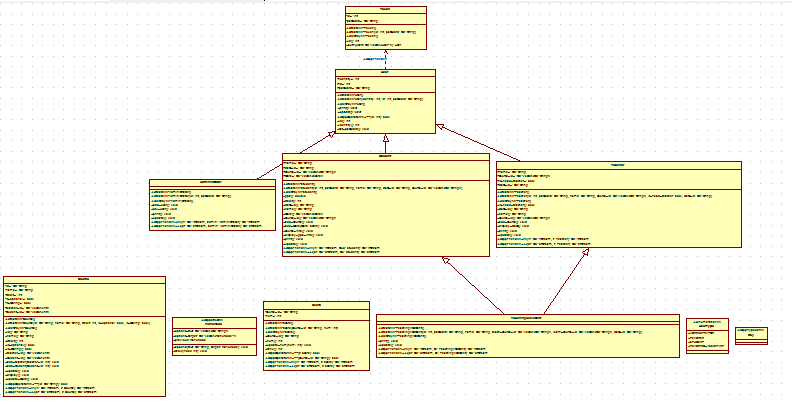


图3.1 UML缩略图

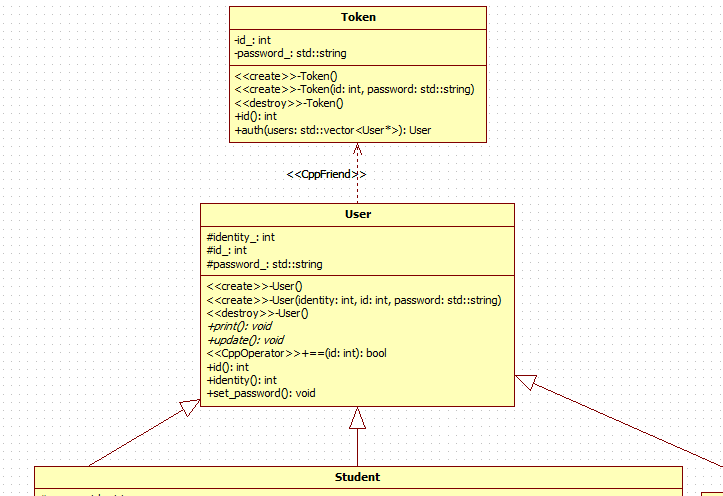


图3.2 UML局部放大图-1

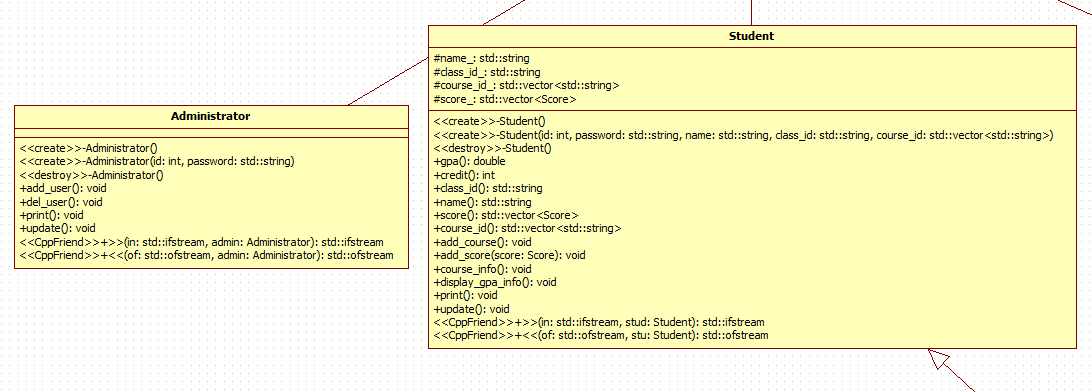


图3.3 UML局部放大图-2

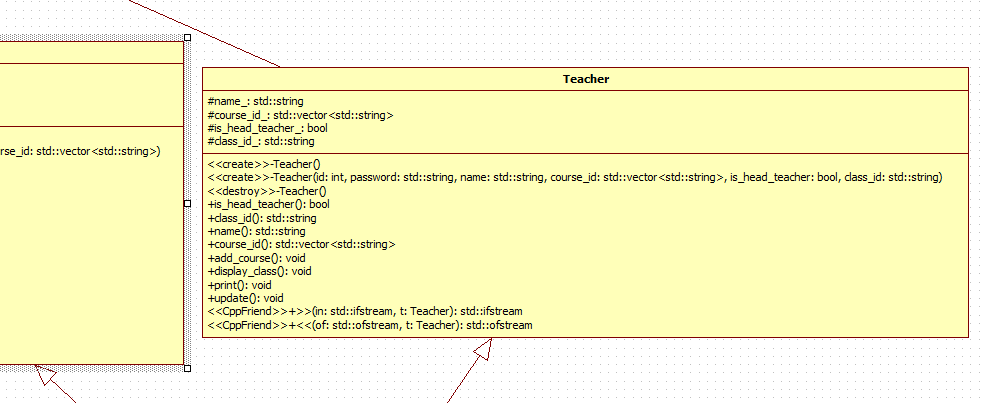


图3.4 UML局部放大图-3

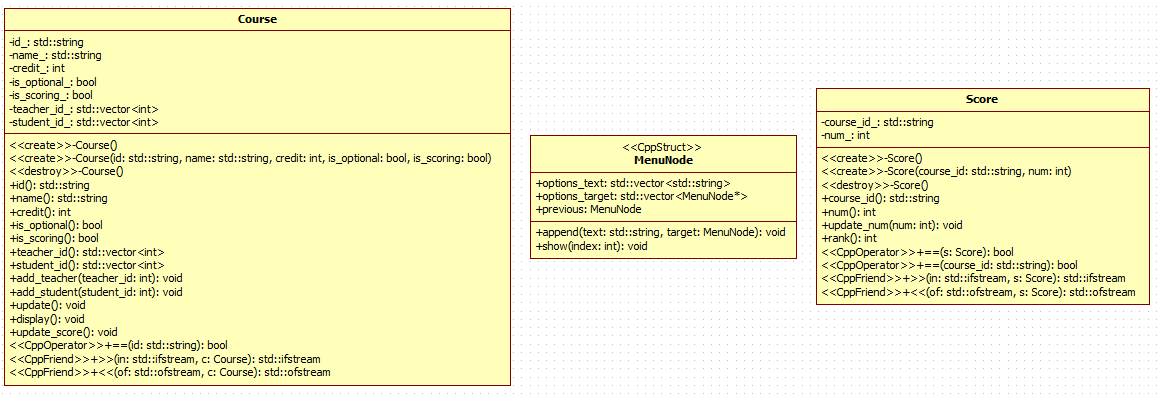


图3.5 UML局部放大图-4

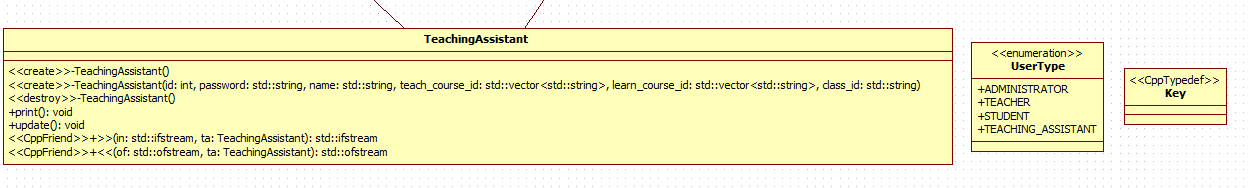


图3.6 UML局部放大图-5

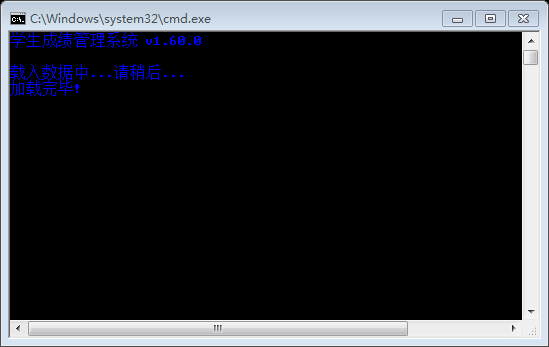
1. 系统调试

将声明部分放入\*.h，函数定义实现放入\*.cpp。编译链接中出现的错误包括粗心导致的变量拼写错误；头文件包含顺序混乱；抽象类不能实例化；‘.’和‘->’操作符的区别；语句过长导致的逻辑混乱；STL容器中迭代器iterator和const\_iterator的用法错误；汉字编码问题；switch后的值必须是整型值；命名冲突；友元函数不会被继承；全局变量的声明（extern）和定义；#ifndef防止头文件重复编译；缺少相应类型的重载函数等。

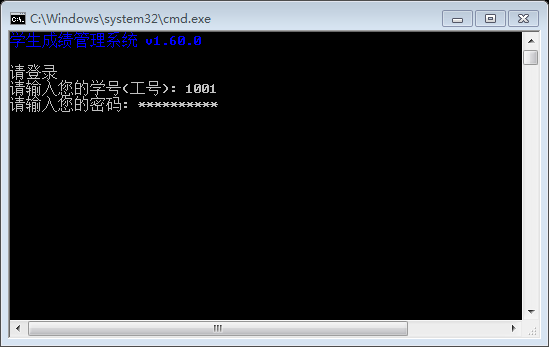
程序编译通过后继续调试，出现的问题包括：许多界面显示结果后直接跳转，导致用户无法看清信息，适时地加入getch()非缓冲不回显的读字符函数用以停留界面；std::cin读取失败后未调用std::cin.sync()清空缓存；密码退格依然以‘\*’回显，加入判断getch()是否为‘\b’并打印相应数量的‘\b’和‘ ’得到解决，之后又加入对左右方向键的相应，密码输入至一半可以利用方向键调整光标位置并保证读入的密码与用户输入意图相同；读取文件到达文件尾导致的错误；用户的不合法输入导致程序进入死循环，原因：实现菜单选项需要获取用户的方向键输入，而一个方向键需要getch()两次，第一次getch()值为224（windows平台），第二次getch()为72,80,75,77（上下左右分别对应），而普通字符只需getch()一次，再getch()第二次则会等待用户输入下一字符，为解决此问题，typedef std::pair<int, char> Key作为新的用户按键类型，自定义MyGetCh()函数利用<windows.h>中的kbhit()函数自动判断应该getch()的次数times，并获取到最后一次getch()的char型值，返回std::make\_pair(times, char)，用MyGetCh()替换原程序中的getch()，成功地解决了用户不合法输入程序不能正常运行的问题，极大地增强了程序的鲁棒性。

1. 测试结果与分析

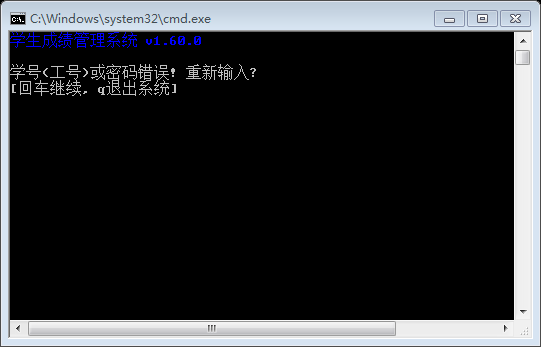
程序运行截图：



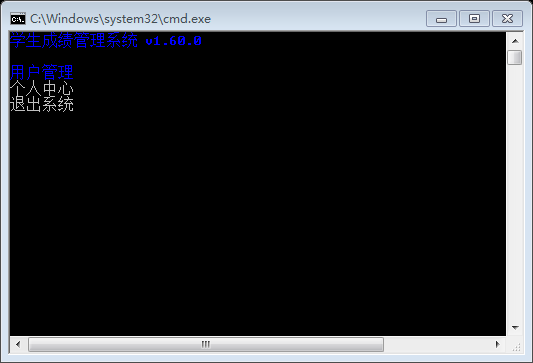
加载数据



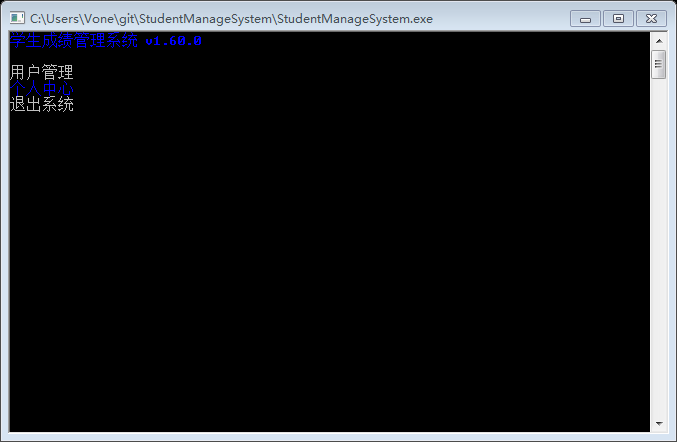
登录界面



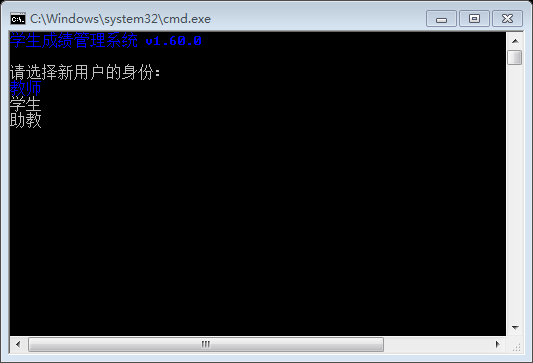
登录失败



菜单界面



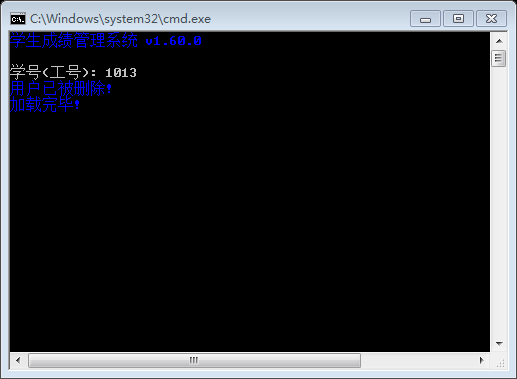
菜单切换选项

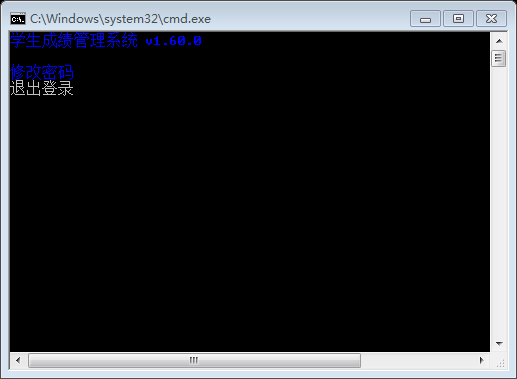


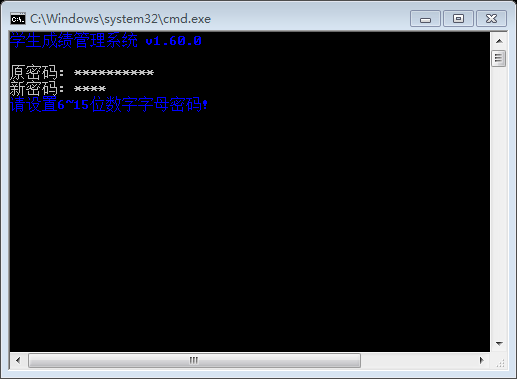
创建新用户



创建教师



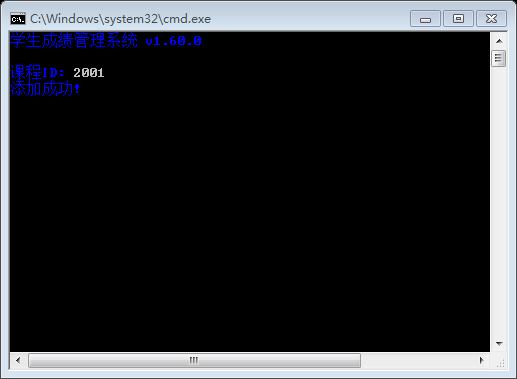
删除用户

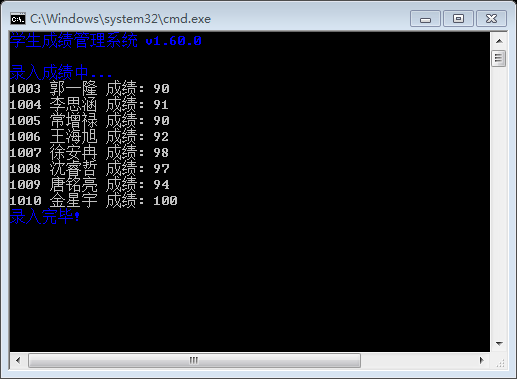


修改密码

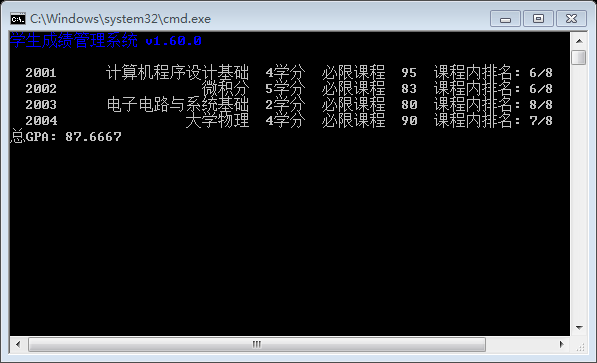


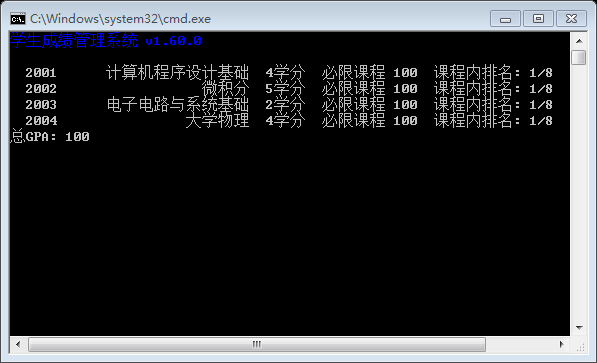
增开新课

学生添加课程

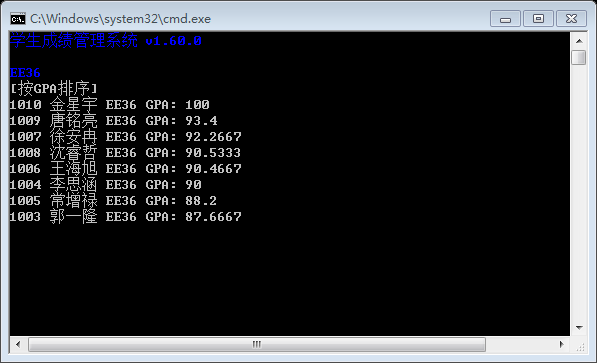


教师录入成绩





学生查看所有课程



班主任查看班级信息

由于删除课程牵连的数据较多，所以实际程序中砍掉了删除课程的功能。

程序特色：

* 1. 密码‘\*’回显，支持退格和方向键，识别F1~F12和全角字符并自动过滤；
  2. 菜单选项高亮显示，接受方向键、回车键和ESC键，其他无效输入自动忽略，调用清屏函数刷新菜单实现舒适的视觉效果。

程序不足：

a) 除了登录界面，后续的操作界面未显示当前操作的用户信息。

b) 排序只有GPA排序一种，应加入学号排序等。

c) 缺少学生总学分的统计。

d) 删除课程功能没有实现。

e) 排版不够美观。

f) 密码明文保存在文件中，非常不安全。

g) 缺少查看所有用户的功能。

h) 默认密码与ID号相同，安全性弱，应随机生成。

1. 总结

这次课设对我写代码能力的锻炼及其巨大！这次写程序试着大量使用STL中的vector,string,pair,map等容器，确实提供了不少便利，调用了windows API的东西实现了一些功能，学习了许多知识。调试过程中，有缺分号少括号这种让人哭笑不得的低级错误，也有“常量中有换行符”这种让人不知所云逼格极高的编码问题，不过可喜的是，最终都通过自己的努力（~~和抱大腿~~）一一解决，非常有成就感！亲手编一个（~~勉强~~）可以用的程序的感觉还是很好的！

1. 附录：源程序清单

// 先贴上主函数，之后头文件源文件按文件名字母顺序依次贴上

// 附几组用户名和密码

// 管理员 1001:2013011189

// 教师 1013:1013

// 教师（班主任）1014:1014

// 学生 1003:1003

// main.cpp

#include "global.h"

#include "file.h"

#include "token.h"

#include "user.h"

#include <conio.h>

#include <fstream>

#include <iostream>

#include <stdlib.h>

#include <map>

int main(int argc, char const \*argv[])

{

login: // 登录标签.

ClearScreen();

HighlightPrint("载入数据中...请稍后...\n");

UpdateUsers();

ClearScreen(); // 保留软件标题的清屏函数.

Token tmp; // 获取登录信息的令牌.

User\* user; // 当前登录的用户.

while (true)

{

tmp = Login(); // 从键盘获取登录信息.

if ((user = tmp.auth(users)) != NULL)

{

std::cout << "登录成功!\n";

std::cout << "欢迎";

user->print(); // 显示用户基本信息.

MyGetCh();

break;

}

else

{

ClearScreen(); // 清屏显示错误.

std::cout << "学号(工号)或密码错误! 重新输入? \n";

std::cout << "[回车继续, q退出系统] ";

if (MyGetCh() == std::make\_pair(1,'q')) // 退出系统.

{

HighlightPrint("\n再见! \n");

return 0;

}

ClearScreen();

}

} // 登录成功.

// 根据用户身份初始化菜单.

switch (user->identity())

{

// 管理员菜单.

case ADMINISTRATOR:

{

Administrator admin;

admin = admins[Find(admins, tmp.id())]; // 从admins中读取当前登录的admin信息.

MenuNode top\_menu, manager\_menu, user\_menu; // 管理员菜单.

top\_menu.previous = manager\_menu.previous = user\_menu.previous = &top\_menu;

top\_menu.append("用户管理", &manager\_menu);

top\_menu.append("个人中心", &user\_menu);

top\_menu.append("退出系统", NULL);

manager\_menu.append("新用户注册", NULL);

manager\_menu.append("删除用户", NULL);

user\_menu.append("修改密码", NULL);

user\_menu.append("退出登录", NULL);

MenuNode now = top\_menu; // 当前显示菜单.

int k = 0; // 菜单项目索引.

while (true)

{

now.show(k); // 展示菜单, 第k项高亮显示

Key ch = MyGetCh(); // 获取键盘输入(非缓冲)

if (ch == UP)

{

if (--k < 0)

k = now.options\_target.size() - 1; // 选中最后一项.

continue;

}

if (ch == DOWN)

{

if (++k > now.options\_target.size() - 1)

k = 0; // 选中第一项.

continue;

}

if (ch == LEFT || ch == ESCAPE)

{

now = \*now.previous; // 返回上级菜单.

k = 0;

continue;

}

if (ch == RIGHT || ch == ENTER)

{

if (now.options\_target[k] == NULL)

{

std::map<std::string, int> mp;

mp["退出系统"] = 0;

mp["新用户注册"] = 1;

mp["删除用户"] = 2;

mp["修改密码"] = 3;

mp["退出登录"] = 4;

switch (mp[now.options\_text[k]])

{

case 0:

{

Exit();

break;

}

case 1:

{

admin.add\_user();

break;

}

case 2:

{

admin.del\_user();

break;

}

case 3:

{

admin.set\_password();

break;

}

case 4:

goto login;

}

k = 0;

continue;

}

else

{

now = \*now.options\_target[k];

k = 0;

continue;

}

}

}

}

// 教师菜单.

case TEACHER:

{

Teacher teacher;

teacher = teachers[Find(teachers, tmp.id())];

MenuNode top\_menu, course\_menu, user\_menu;

top\_menu.previous = course\_menu.previous = user\_menu.previous = &top\_menu;

top\_menu.append("课程管理", &course\_menu);

if (teacher.is\_head\_teacher())

top\_menu.append("查看班级", NULL);

top\_menu.append("个人中心", &user\_menu);

top\_menu.append("退出系统", NULL);

for (int i = 0; i < teacher.course\_id().size(); i++)

course\_menu.append(courses[Find(courses, teacher.course\_id()[i])].name(), NULL);

course\_menu.append("增开课程", NULL);

user\_menu.append("修改密码", NULL);

user\_menu.append("退出登录", NULL);

MenuNode now = top\_menu;

int k = 0;

while (true)

{

now.show(k); // 展示菜单, 第k项高亮显示

Key ch = MyGetCh(); // 获取键盘输入(非缓冲)

if (ch == UP)

{

if (--k < 0)

k = now.options\_target.size() - 1; // 选中最后一项.

continue;

}

if (ch == DOWN)

{

if (++k > now.options\_target.size() - 1)

k = 0; // 选中第一项.

continue;

}

if (ch == LEFT || ch == ESCAPE)

{

now = \*now.previous; // 返回上级菜单.

k = 0;

continue;

}

if (ch == RIGHT || ch == ENTER)

{

if (now.options\_target[k] == NULL)

{

std::map<std::string, int> mp;

mp["退出系统"] = 0;

mp["查看班级"] = 1;

mp["增开课程"] = 2;

mp["修改密码"] = 3;

mp["退出登录"] = 4;

for (int i = 0; i < teacher.course\_id().size(); i++)

mp[courses[Find(courses, teacher.course\_id()[i])].name()] = 5;

switch (mp[now.options\_text[k]])

{

case 0:

{

Exit();

break;

}

case 1:

{

teacher.display\_class();

break;

}

case 2:

{

teacher.add\_course();

break;

}

case 3:

{

teacher.set\_password();

break;

}

case 4:

goto login;

case 5:

{

Course course = courses[Find(courses, now.options\_text[k])];

if (course.is\_scoring())

{

ClearScreen();

std::cout << "现在录入成绩? [y/n]";

ch = MyGetCh();

if (ch == std::make\_pair(1, 'y') || ch == std::make\_pair(1, 'Y'))

{

course.update\_score();

break;

}

}

course.display();

break;

}

}

k = 0;

continue;

}

else

{

now = \*now.options\_target[k];

k = 0;

continue;

}

}

}

}

// 学生菜单.

case STUDENT:

{

Student student;

student = students[Find(students, tmp.id())];

MenuNode top\_menu, user\_menu;

top\_menu.previous = user\_menu.previous = &top\_menu;

top\_menu.append("所有课程", NULL);

top\_menu.append("增加课程", NULL);

top\_menu.append("个人中心", &user\_menu);

top\_menu.append("退出系统", NULL);

user\_menu.append("修改密码", NULL);

user\_menu.append("退出登录", NULL);

MenuNode now = top\_menu;

int k = 0;

while (true)

{

now.show(k); // 展示菜单, 第k项高亮显示

Key ch = MyGetCh(); // 获取键盘输入(非缓冲)

if (ch == UP)

{

if (--k < 0)

k = now.options\_target.size() - 1; // 选中最后一项.

continue;

}

if (ch == DOWN)

{

if (++k > now.options\_target.size() - 1)

k = 0; // 选中第一项.

continue;

}

if (ch == LEFT || ch == ESCAPE)

{

now = \*now.previous; // 返回上级菜单.

k = 0;

continue;

}

if (ch == RIGHT || ch == ENTER)

{

if (now.options\_target[k] == NULL)

{

std::map<std::string, int> mp;

mp["所有课程"] = 0;

mp["增加课程"] = 1;

mp["退出系统"] = 2;

mp["修改密码"] = 3;

mp["退出登录"] = 4;

switch (mp[now.options\_text[k]])

{

case 0:

{

student.course\_info();

break;

}

case 1:

{

student.add\_course();

break;

}

case 2:

{

Exit();

break;

}

case 3:

{

student.set\_password();

break;

}

case 4:

goto login;

}

k = 0;

continue;

}

else

{

now = \*now.options\_target[k];

k = 0;

continue;

}

}

}

}

// 助教菜单

case TEACHING\_ASSISTANT:

{

TeachingAssistant ta;

ta = tas[Find(tas, tmp.id())];

MenuNode top\_menu, teacher\_menu, student\_menu, user\_menu;

top\_menu.previous = teacher\_menu.previous = student\_menu.previous = user\_menu.previous = &top\_menu;

top\_menu.append("教师界面", &teacher\_menu);

top\_menu.append("学生界面", &student\_menu);

top\_menu.append("个人中心", &user\_menu);

top\_menu.append("退出系统", NULL);

for (int i = 0; i < ta.Teacher::course\_id().size(); i++)

teacher\_menu.append(courses[Find(courses, ta.Teacher::course\_id()[i])].name(), NULL);

teacher\_menu.append("增开课程", NULL);

student\_menu.append("所有课程", NULL);

student\_menu.append("增加课程", NULL);

user\_menu.append("修改密码", NULL);

user\_menu.append("退出登录", NULL);

MenuNode now = top\_menu;

int k = 0;

while (true)

{

now.show(k); // 展示菜单, 第k项高亮显示

Key ch = MyGetCh(); // 获取键盘输入(非缓冲)

if (ch == UP)

{

if (--k < 0)

k = now.options\_target.size() - 1; // 选中最后一项.

continue;

}

if (ch == DOWN)

{

if (++k > now.options\_target.size() - 1)

k = 0; // 选中第一项.

continue;

}

if (ch == LEFT || ch == ESCAPE)

{

now = \*now.previous; // 返回上级菜单.

k = 0;

continue;

}

if (ch == RIGHT || ch == ENTER)

{

if (now.options\_target[k] == NULL)

{

std::map<std::string, int> mp;

mp["退出系统"] = 0;

mp["增开课程"] = 1;

mp["所有课程"] = 2;

mp["增加课程"] = 3;

mp["修改密码"] = 4;

mp["退出登录"] = 5;

for (int i = 0; i < ta.Teacher::course\_id().size(); i++)

mp[courses[Find(courses, ta.Teacher::course\_id()[i])].name()] = 6;

switch (mp[now.options\_text[k]])

{

case 0:

{

Exit();

break;

}

case 1:

{

ta.Teacher::add\_course();

break;

}

case 2:

{

ta.Student::course\_info();

break;

}

case 3:

{

ta.Student::add\_course();

break;

}

case 4:

{

ta.set\_password();

break;

}

case 5:

goto login;

case 6:

{

Course course = courses[Find(courses, now.options\_text[k])];

if (course.is\_scoring())

{

ClearScreen();

std::cout << "现在录入成绩? [y/n]";

ch = MyGetCh();

if (ch == std::make\_pair(1, 'y') || ch == std::make\_pair(1, 'Y'))

{

course.update\_score();

break;

}

}

course.display();

break;

}

}

k = 0;

continue;

}

else

{

now = \*now.options\_target[k];

k = 0;

continue;

}

}

}

}

}

return 0;

}

#ifndef COURSE\_H

#define COURSE\_H

#include <string>

#include <vector>

const int kCourseMaxSize = 400; // 课容量最大值

const int kCourseMaxTeacher = 5; // 课程老师最大值(包括主讲老师,助教等)

class Course

{

public:

Course():id\_(), name\_(), credit\_(), is\_optional\_(), is\_scoring\_() {}

Course(std::string id, std::string name, int credit, bool is\_optional, bool is\_scoring):

id\_(id), name\_(name), credit\_(credit), is\_optional\_(is\_optional), is\_scoring\_(is\_scoring) {}

~Course() {}

std::string id(){ return id\_; }

std::string name(){ return name\_; }

int credit(){ return credit\_; }

bool is\_optional(){ return is\_optional\_; }

bool is\_scoring(){ return is\_scoring\_; }

std::vector<int> teacher\_id(){ return teacher\_id\_; }

std::vector<int> student\_id(){ return student\_id\_; }

void add\_teacher(int teacher\_id);

void add\_student(int student\_id);

void update();

void display();

void update\_score();

bool operator ==(std::string &id){ return ((id\_ == id) || (name\_ == id)); }

friend std::ifstream &operator >>(std::ifstream &in, Course &c);

friend std::ofstream &operator <<(std::ofstream &of, const Course &c);

private:

std::string id\_; // 课程ID

std::string name\_; // 课程名称

int credit\_; // 学分

bool is\_optional\_; // 是否任选(非任选则为必修限选课程)

bool is\_scoring\_; // 是否记分

std::vector<int> teacher\_id\_; // 老师团队

std::vector<int> student\_id\_; // 学生ID

};

#endif

// file.h

// 声明文件操作所需的函数

#ifndef FILE\_H

#define FILE\_H

// user info

void WriteAdmins();

void WriteTeachers();

void WriteStudents();

void WriteTAs();

void ReadAdmins();

void ReadTeachers();

void ReadStudents();

void ReadTAs();

void ReadUsers();

void WriteCourses();

void ReadCourses();

#endif

// global.h

#ifndef GLOBAL\_H

#define GLOBAL\_H

#include "user.h"

#include <string>

#include <utility>

#include "course.h"

#include "token.h"

#include "menu.h"

extern std::vector<User\*> users; // 所有用户基类指针.

extern std::vector<Administrator> admins;

extern std::vector<Teacher> teachers;

extern std::vector<Student> students;

extern std::vector<TeachingAssistant> tas; // ta: teachingassistant

extern std::vector<Course> courses;

typedef std::pair<int, char> Key; // first = times, second = ch;

const Key ENTER = std::make\_pair(1, char(13));

const Key ESCAPE = std::make\_pair(1, char(27));

const Key UP = std::make\_pair(2, char(72));

const Key DOWN = std::make\_pair(2, char(80));

const Key LEFT = std::make\_pair(2, char(75));

const Key RIGHT = std::make\_pair(2, char(77));

Key MyGetCh();

std::string GetPass();

Token Login();

void ClearScreen();

void Exit();

void Title();

void UpdateFiles();

void UpdateUsers();

int Find(std::vector<User\*> v, int id);

template <class A, class B>

int Find(std::vector<A> v, B id)

{

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

{

if (v[i] == id)

return i;

}

return -1;

}

template <class A, class B>

std::vector<A> Remove(std::vector<A> v, B id)

{

for (std::vector<A>::iterator it = v.begin(); it != v.end(); it++)

{

if (\*it == id)

{

v.erase(it);

break;

}

}

return v;

}

#endif

// menu.h

#ifndef MENU\_H

#define MENU\_H

#include <iostream>

#include <conio.h>

#include <string>

#include <windows.h>

#include <vector>

bool HighlightPrint(std::string text);

struct MenuNode

{

std::vector<std::string> options\_text;

std::vector<MenuNode\*> options\_target;

MenuNode\* previous;

void append(std::string text, MenuNode\* target);

void show(int index);

};

#endif

#ifndef SCORE\_H

#define SCORE\_H

#include <string>

class Score

{

public:

Score() {}

Score(std::string course\_id, int num): course\_id\_(course\_id), num\_(num) {};

~Score() {}

std::string course\_id() { return course\_id\_; }

int num() { return num\_; }

void update\_num(int num) { num\_ = num; }

int rank();

bool operator ==(const Score &s) { return course\_id\_ == s.course\_id\_; }

bool operator ==(const std::string &course\_id) { return course\_id\_ == course\_id; }

friend std::ifstream &operator >>(std::ifstream &in, Score &s);

friend std::ofstream &operator <<(std::ofstream &of, const Score &s);

private:

std::string course\_id\_; // 课程ID

int num\_; // 分数

};

bool CompareScore(Score s1, Score s2);

#endif

// token.h

#ifndef TOKEN\_H

#define TOKEN\_H

#include <string>

#include <vector>

#include "user.h"

class Token

{

public:

Token() {}

Token(int id, std::string password): id\_(id), password\_(password) {}

~Token() {}

int id() { return id\_; }

User\* auth(const std::vector<User\*> users);

private:

int id\_;

std::string password\_;

};

#endif

// user.h

// 抽象类User派生出Administrator,Teacher,Student类,TeachingAssistant类多重继承自Teacher类和Student类

#ifndef USER\_H

#define USER\_H

#include <string>

#include <vector>

#include <iostream>

#include <fstream>

#include "score.h"

enum UserType { ADMINISTRATOR, TEACHER, STUDENT, TEACHING\_ASSISTANT };

// 用户类型分为管理员,老师,学生,助教四种身份.

bool ValidPassword(std::string password); // 判断密码是否符合标准.(数字,大小写字母的6-15位组合)

class User

{

public:

friend class Token;

User() {}

User(int identity, int id, std::string password): identity\_(identity), id\_(id), password\_(password) {}

virtual ~User() {}

virtual void print() = 0;

virtual void update() = 0; // 更新全局vector

bool operator ==(const int id){ return id\_ == id; }

int id() { return id\_; }

int identity() { return identity\_; }

void set\_password();

protected:

int identity\_; // 用户身份

int id\_; // 用户ID

std::string password\_; // 登录管理系统所用密码

};

class Administrator: public User // 继承自user抽象类

{

public:

Administrator() {}

Administrator(int id, std::string password): User(ADMINISTRATOR, id, password){}

~Administrator() {}

void add\_user(); // 增加新用户

void del\_user(); // 删除用户

void print() { std::cout << "管理员: " << id\_ << std::endl; }

void update();

friend std::ifstream &operator >>(std::ifstream &in, Administrator &admin);

friend std::ofstream &operator <<(std::ofstream &of, const Administrator &admin);

};

class Teacher: virtual public User

{

public:

Teacher():User(), name\_(), course\_id\_(), is\_head\_teacher\_(0), class\_id\_() { course\_id\_.reserve(100); }

Teacher(int id, std::string password, std::string name, std::vector<std::string> course\_id, bool is\_head\_teacher = 0, std::string class\_id = 0):

User(TEACHER, id, password), name\_(name), course\_id\_(course\_id), is\_head\_teacher\_(is\_head\_teacher), class\_id\_(class\_id) {}

~Teacher() {}

bool is\_head\_teacher() { return is\_head\_teacher\_; }

std::string class\_id() { return class\_id\_; }

std::string name() { return name\_; }

std::vector<std::string> course\_id() const { return course\_id\_; }

void add\_course();

void display\_class();

void print() { std::cout << name\_ << "老师" << std::endl; }

void update();

friend std::ifstream &operator >>(std::ifstream &in, Teacher &t);

friend std::ofstream &operator <<(std::ofstream &of, const Teacher &t);

protected:

std::string name\_; // 老师姓名

std::vector<std::string> course\_id\_; // 课程ID

bool is\_head\_teacher\_; // 是否班主任

std::string class\_id\_; // 若是班主任,班级ID

};

const int kStudentMaxCredit = 32; // 单学期学分上限

class Student: virtual public User

{

public:

Student():User(), name\_(), class\_id\_(), course\_id\_(), score\_() { course\_id\_.reserve(100), score\_.reserve(100); }

Student(int id, std::string password, std::string name, std::string class\_id,

std::vector<std::string> course\_id):

User(STUDENT, id, password), name\_(name), class\_id\_(class\_id), course\_id\_(course\_id) {}

~Student() {}

double gpa();

int credit();

std::string class\_id() { return class\_id\_; }

std::string name() { return name\_; }

std::vector<Score> score() { return score\_; }

std::vector<std::string> course\_id() const { return course\_id\_; }

void add\_course();

void add\_score(Score score);

void course\_info();

void display\_gpa\_info();

void print() { std::cout << name\_ << "同学 " << '(' << id\_ << ')' << std::endl; }

void update();

friend std::ifstream &operator >>(std::ifstream &in, Student &stud);

friend std::ofstream &operator <<(std::ofstream &of, const Student &stu);

protected:

std::string name\_; // 学生姓名

std::string class\_id\_; // 班号

std::vector<std::string> course\_id\_; // 课程ID

std::vector<Score> score\_; // 各课程分数

};

bool CompareStudent(Student stu1, Student stu2);

class TeachingAssistant: public Teacher, public Student

{

public:

TeachingAssistant():User(), Teacher(), Student() {}

TeachingAssistant(int id, std::string password, std::string name, std::vector<std::string> teach\_course\_id,

std::vector<std::string> learn\_course\_id, std::string class\_id):

User(TEACHING\_ASSISTANT, id, password), Teacher(id, password, name, teach\_course\_id),

Student(id, password, name, class\_id, learn\_course\_id) {}

~TeachingAssistant() {}

void print() { std::cout << Teacher::name() << "助教 " << std::endl; }

void update();

friend std::ifstream &operator >>(std::ifstream &in, TeachingAssistant &ta);

friend std::ofstream &operator <<(std::ofstream &of, const TeachingAssistant &ta);

};

#endif

// course.cpp

#include "course.h"

#include "user.h"

#include "file.h"

#include "global.h"

void Course::add\_teacher(int teacher\_id)

{

teacher\_id\_.push\_back(teacher\_id);

update();

return;

}

void Course::add\_student(int student\_id)

{

student\_id\_.push\_back(student\_id);

update();

return;

}

void Course::update()

{

courses = Remove(courses, id\_);

courses.push\_back(\*this);

WriteCourses();

return;

}

void Course::display()

{

ClearScreen();

std::cout << id\_ << ' ';

std::cout << name\_ << ' ';

std::cout << credit\_ << "学分 ";

if (!is\_scoring\_)

std::cout << "不记分 ";

if (is\_optional\_)

std::cout << "选修课程 \n";

else

std::cout << "必限课程 \n";

HighlightPrint("讲师: \n");

for (std::vector<int>::iterator it = teacher\_id\_.begin(); it != teacher\_id\_.end(); it++)

{

if (Find(teachers, \*it) < 0)

tas[Find(tas, \*it)].print();

else

teachers[Find(teachers, \*it)].print();

}

HighlightPrint("学生: \n");

for (std::vector<int>::iterator it = student\_id\_.begin(); it != student\_id\_.end(); it++)

{

Student stu = students[Find(students, \*it)];

if (!is\_scoring\_ || Find(stu.score(), id\_) < 0)

stu.print();

else

{

std::cout << stu.id() << ' ';

std::cout << stu.name() << ' ';

std::cout << stu.score()[Find(stu.score(), id\_)].num() << std::endl;

}

}

MyGetCh();

return;

}

void Course::update\_score()

{

ClearScreen();

HighlightPrint("录入成绩中...\n");

for (std::vector<int>::iterator it = student\_id\_.begin(); it != student\_id\_.end(); it++)

{

Student stu = students[Find(students, \*it)];

std::cout << stu.id() << ' ';

std::cout << stu.name() << ' ';

std::cout << "成绩: ";

int num = 0;

if (!(std::cin >> num) || num < 0 || num > 100)

{

std::cin.clear();

std::cin.sync();

HighlightPrint("输入错误!\n");

MyGetCh();

return;

}

std::cin.get();

stu.add\_score(Score(id\_, num));

stu.update();

}

HighlightPrint("录入完毕!\n");

MyGetCh();

return;

}

std::ifstream &operator >>(std::ifstream &in, Course &c)

{

in >> c.id\_;

in >> c.name\_;

in >> c.credit\_;

in >> c.is\_optional\_;

in >> c.is\_scoring\_;

char start\_flag;

while ((start\_flag = in.get()) == '\n')

;

if (start\_flag == '\*')

{

while (true)

{

char end\_flag;

while ((end\_flag = in.get()) == '\n')

;

if (end\_flag == '#')

break;

else

{

in.seekg(-1, std::ios::cur);

int tmp\_id;

in >> tmp\_id;

c.teacher\_id\_.push\_back(tmp\_id);

}

}

}

else

in.seekg(-1, std::ios::cur);

while ((start\_flag = in.get()) == '\n')

;

if (start\_flag == '\*')

{

while (true)

{

char end\_flag;

while ((end\_flag = in.get()) == '\n')

;

if (end\_flag == '#')

break;

else

{

in.seekg(-1, std::ios::cur);

int tmp\_id;

in >> tmp\_id;

c.student\_id\_.push\_back(tmp\_id);

}

}

}

else

in.seekg(-1, std::ios::cur);

return in;

}

std::ofstream &operator <<(std::ofstream &of, const Course &c)

{

of << c.id\_ << '\n';

of << c.name\_ << '\n';

of << c.credit\_ << '\n';

of << c.is\_optional\_ << '\n';

of << c.is\_scoring\_ << '\n';

of << "\*\n";

for (std::vector<int>::const\_iterator it = c.teacher\_id\_.begin(); it != c.teacher\_id\_.end(); it++)

of << \*it << '\n';

of << "#\n";

of << "\*\n";

for (std::vector<int>::const\_iterator it = c.student\_id\_.begin(); it != c.student\_id\_.end(); it++)

of << \*it << '\n';

of << "#\n\n";

return of;

}

// file.cpp

#include "file.h"

#include "global.h"

#include "user.h"

#include <fstream>

void WriteAdmins()

{

std::ofstream of("./data/admins.txt", std::ios::out|std::ios::trunc);

for (std::vector<Administrator>::iterator it = admins.begin(); it != admins.end(); it++)

of << \*it;

}

void WriteTeachers()

{

std::ofstream of("./data/teachers.txt", std::ios::out|std::ios::trunc);

for (std::vector<Teacher>::iterator it = teachers.begin(); it != teachers.end(); it++)

of << \*it;

}

void WriteStudents()

{

std::ofstream of("./data/students.txt", std::ios::out|std::ios::trunc);

for (std::vector<Student>::iterator it = students.begin(); it != students.end(); it++)

of << \*it;

}

void WriteTAs()

{

std::ofstream of("./data/tas.txt", std::ios::out|std::ios::trunc);

for (std::vector<TeachingAssistant>::iterator it = tas.begin(); it != tas.end(); it++)

of << \*it;

}

void ReadAdmins()

{

std::ifstream in("./data/admins.txt", std::ios::in);

admins.clear();

Administrator admin;

while (in >> admin)

{

admins.push\_back(admin);

admin = Administrator();

}

return;

}

void ReadTeachers()

{

std::ifstream in("./data/teachers.txt", std::ios::in);

teachers.clear();

Teacher teacher;

while (in >> teacher)

{

teachers.push\_back(teacher);

teacher = Teacher();

}

return;

}

void ReadStudents()

{

std::ifstream in("./data/students.txt", std::ios::in);

students.clear();

Student student;

while (in >> student)

{

students.push\_back(student);

student = Student();

}

return;

}

void ReadTAs()

{

std::ifstream in("./data/tas.txt", std::ios::in);

tas.clear();

TeachingAssistant ta;

while (in >> ta)

{

tas.push\_back(ta);

ta = TeachingAssistant();

}

return;

}

void ReadUsers()

{

users.clear();

User\* up;

for (std::vector<Administrator>::iterator it = admins.begin(); it != admins.end(); it++)

{

up = &(\*it);

users.push\_back(up);

}

for (std::vector<Teacher>::iterator it = teachers.begin(); it != teachers.end(); it++)

{

up = &(\*it);

users.push\_back(up);

}

for (std::vector<Student>::iterator it = students.begin(); it != students.end(); it++)

{

up = &(\*it);

users.push\_back(up);

}

for (std::vector<TeachingAssistant>::iterator it = tas.begin(); it != tas.end(); it++)

{

up = &(\*it);

users.push\_back(up);

}

return;

}

void WriteCourses()

{

std::ofstream of("./data/courses.txt", std::ios::out|std::ios::trunc);

for (std::vector<Course>::iterator it = courses.begin(); it != courses.end(); it++)

of << \*it;

}

void ReadCourses()

{

std::ifstream in("./data/courses.txt", std::ios::in);

courses.clear();

Course tmp;

while (in >> tmp)

{

courses.push\_back(tmp);

tmp = Course();

}

}

// global.cpp

#include "global.h"

#include "course.h"

#include "file.h"

std::vector<User\*> users; // 所有用户基类指针.

std::vector<Administrator> admins;

std::vector<Teacher> teachers;

std::vector<Student> students;

std::vector<TeachingAssistant> tas; // ta: teachingassistant

std::vector<Course> courses;

Key MyGetCh()

{

int ch = getch();

if (kbhit())

return std::make\_pair(2, getch());

return std::make\_pair(1, ch);

}

std::string GetPass()

{

int c;

std::string password;

std::string::iterator it = password.end();

while ((c = getch()) != '\r')

{

switch (c)

{

case 0:

{

getch();

std::cout << '\a';

break;

}

case 224:

{

switch (getch())

{

case 75: // left

{

if (it != password.begin())

{

--it;

std::cout << '\b';

}

else

std::cout << '\a';

break;

}

case 77: // right

{

if (it != password.end())

{

++it;

std::cout << '\*';

}

else

std::cout << '\a';

break;

}

default:

{

std::cout << '\a';

break;

}

}

break;

}

case 163:

{

getch();

std::cout << '\a';

break;

}

case '\b':

{

if (password.size() != 0 && it != password.begin())

{

std::cout << '\b';

for (int i = 0; i < password.end() - it; ++i)

std::cout << '\*';

std::cout << ' ';

for (int i = 0; i < password.end() - it + 1; ++i)

std::cout << '\b';

password.erase(it - 1);

--it;

}

else

std::cout << '\a';

break;

}

default:

{

if (isalnum(c) || ispunct(c))

{

password.insert(it, c);

++it;

for (int i = 0; i < password.end() - it + 1; ++i)

std::cout << '\*';

for (int i = 0; i < password.end() - it; ++i)

std::cout << '\b';

}

else

std::cout << '\a';

break;

}

}

}

std::cout << std::endl;

return password;

}

Token Login()

{

ClearScreen();

std::cout << "请登录\n";

int id;

std::string pwd;

std::cout << "请输入您的学号(工号): ";

if (!(std::cin >> id))

{

std::cin.clear();

std::cin.sync();

return Token(-1, "");

}

std::cin.get();

std::cout << "请输入您的密码: ";

pwd = GetPass();

return Token(id, pwd);

}

void ClearScreen()

{

system("cls");

Title();

}

void Exit()

{

ClearScreen(); // 清屏

std::cout << "确认退出? [y/n]" << std::endl;

Key ch = MyGetCh();

if (ch == std::make\_pair(1, 'y') || ch == std::make\_pair(1, 'Y'))

{

std::cout << "再见!" << std::endl;

exit(1);

}

}

void Title()

{

HighlightPrint("学生成绩管理系统 v1.60.0\n\n");

}

void UpdateFiles()

{

// 根据全局vector更新文件.

WriteAdmins();

WriteTeachers();

WriteStudents();

WriteTAs();

WriteCourses();

}

void UpdateUsers()

{

// 从文件读取用户信息.

ReadAdmins();

ReadTeachers();

ReadStudents();

ReadTAs();

ReadUsers();

ReadCourses();

HighlightPrint("加载完毕! \n");

MyGetCh();

return;

}

int Find(std::vector<User\*> v, int id)

{

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

{

if (v[i]->id() == id)

return i;

}

return -1;

}

// menu.cpp

#include "menu.h"

#include "global.h"

bool HighlightPrint(std::string text)

{

HANDLE handle = GetStdHandle(STD\_OUTPUT\_HANDLE);

if (handle == 0)

return false;

BOOL ret = SetConsoleTextAttribute(handle, WORD(9));

std::cout << text;

ret = SetConsoleTextAttribute(handle, WORD(7));

return (ret == true);

}

void MenuNode::append(std::string text, MenuNode\* target)

{

options\_text.push\_back(text);

options\_target.push\_back(target);

}

void MenuNode::show(int index)

{

ClearScreen();

for (int i = 0; i < options\_text.size(); i++)

{

if (i == index)

{

HighlightPrint(options\_text[i] + '\n');

continue;

}

std::cout << options\_text[i] << std::endl;

}

}

//score.cpp

#include "score.h"

#include "global.h"

#include <algorithm>

bool CompareScore(Score s1, Score s2)

{

return s1.num() > s2.num();

}

int Score::rank()

{

Course course = courses[Find(courses, course\_id\_)];

std::vector<Score> scores;

std::vector<int> student\_id = course.student\_id();

for (std::vector<int>::const\_iterator it = student\_id.begin(); it != student\_id.end(); it++)

{

Student stu = students[Find(students, \*it)];

scores.push\_back(stu.score()[Find(stu.score(), course\_id\_)]);

}

std::sort(scores.begin(), scores.end(), CompareScore);

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

{

if (num\_ == scores[i].num())

return i + 1;

}

return 0;

}

std::ifstream &operator >>(std::ifstream &in, Score &s)

{

in >> s.course\_id\_;

in >> s.num\_;

return in;

}

std::ofstream &operator <<(std::ofstream &of, const Score &s)

{

of << s.course\_id\_ << '\n';

of << s.num\_ << '\n';

return of;

}

// token.cpp

#include "token.h"

#include <iostream>

User\* Token::auth(const std::vector<User\*> users)

{

std::vector<User\*>::const\_iterator it;

for (it = users.begin(); it != users.end(); it++)

{

if ((\*\*it).id\_ == id\_)

{

if ((\*\*it).password\_ == password\_)

return \*it;

else

return NULL; // Invalid password.

}

}

return NULL; // User does not exist.

}

// user.cpp

#include "user.h"

#include "course.h"

#include "global.h"

#include <algorithm>

#include "file.h"

bool ValidPassword(std::string password)

{

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

{

char ch = password[i];

if (!((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z') || (ch >= '0' && ch <= '9')))

return false;

}

return ((password.size() >= 6) && (password.size() <= 15));

}

void User::set\_password()

{

ClearScreen();

std::cout << "原密码: ";

std::string origin = GetPass();

if (origin != password\_)

{

HighlightPrint("密码错误! \n");

MyGetCh();

return;

}

std::cout << "新密码: ";

std::string new\_pwd = GetPass();

if (!ValidPassword(new\_pwd))

{

HighlightPrint("请设置6~15位数字字母密码!\n");

MyGetCh();

return;

}

std::cout << "重复新密码: ";

std::string confirm = GetPass();

if (new\_pwd != confirm)

{

HighlightPrint("两次密码不一致!\n");

MyGetCh();

return;

}

password\_ = new\_pwd;

update();

HighlightPrint("设置成功!\n");

MyGetCh();

return;

}

void Administrator::add\_user()

{

std::vector<std::string> identities;

int k = 1; // identity = k

identities.push\_back("教师");

identities.push\_back("学生");

identities.push\_back("助教");

while (true) // 身份菜单选择.

{

ClearScreen();

std::cout << "请选择新用户的身份:\n";

for (int i = 0; i < 3; ++i)

{

if (i == k - 1)

{

HighlightPrint(identities[i] + '\n');

continue;

}

std::cout << identities[i] + '\n';

}

Key ch = MyGetCh();

if (ch == UP)

{

if (--k < 1)

k = 3;

continue;

}

if (ch == DOWN)

{

if (++k > 3)

k = 1;

continue;

}

if (ch == LEFT || ch == ESCAPE)

return;

if (ch == RIGHT || ch == ENTER)

{

ClearScreen();

HighlightPrint(identities[k - 1] + '\n');

continue;

}

}

int new\_id = 0;

for (std::vector<User\*>::iterator it = users.begin(); it != users.end(); it++)

{

if (new\_id <= (\*it)->id())

new\_id = (\*it)->id() + 1;

}

std::cout << "学号(工号): " << new\_id << std::endl;

std::cout << "请输入姓名: ";

std::string new\_name;

std::cin >> new\_name;

std::cin.get();

std::cout << "密码单打印完毕!\n"; // 默认密码与学号(工号)相同

HighlightPrint("请尽快修改密码!\n");

switch (k)

{

case TEACHER:

{

std::ofstream fp("./data/teachers.txt", std::ios::app|std::ios::out);

fp << k << '\n';

fp << new\_id << '\n';

fp << new\_id << '\n'; // 密码

fp << new\_name << '\n';

fp << "\*\n";

fp << "#\n";

std::cout << "若新用户为班主任, 请输入班号: [否则请直接回车] ";

if (MyGetCh() != ENTER) // 班主任

{

fp << 1 << '\n';

std::string class\_id;

std::cin >> class\_id;

std::cin.get();

fp << class\_id << "\n\n";

fp.close();

break;

}

else // 非班主任

{

fp << 0 << '\n';

fp << 0 << "\n\n";

fp.close();

break;

}

}

case STUDENT:

{

std::ofstream fp("./data/students.txt", std::ios::app|std::ios::out);

fp << k << '\n';

fp << new\_id << '\n';

fp << new\_id << '\n'; // 密码

fp << new\_name << '\n';

std::cout << "班号: ";

std::string class\_id;

std::cin >> class\_id;

std::cin.get();

fp << class\_id << '\n';

fp << "\*\n";

fp << "#\n";

fp << "\*\n";

fp << "#\n";

fp.close();

break;

}

case TEACHING\_ASSISTANT:

{

std::ofstream fp("./data/tas.txt", std::ios::app|std::ios::out);

fp << k << '\n';

fp << new\_id << '\n';

fp << new\_id << '\n'; // 密码

fp << new\_name << '\n';

std::cout << "班号: ";

std::string class\_id;

std::cin >> class\_id;

std::cin.get();

fp << class\_id << '\n';

fp << "\*\n";

fp << "#\n";

fp << "\*\n";

fp << "#\n";

fp.close();

break;

}

default:

break;

}

UpdateUsers();

return;

}

void Administrator::del\_user()

{

ClearScreen();

std::cout << "学号(工号): ";

int del\_id;

std::cin >> del\_id;

std::cin.get();

if (Find(users, del\_id) < 0) // 用户不存在

{

std::cout << "用户不存在!\n";

MyGetCh();

ClearScreen();

return;

}

switch (users[Find(users, del\_id)]->identity())

{

case ADMINISTRATOR:

{

HighlightPrint("无权限进行此操作! \n");

break;

}

case TEACHER:

{

teachers = Remove(teachers, del\_id);

HighlightPrint("用户已被删除!\n");

break;

}

case STUDENT:

{

students = Remove(students, del\_id);

HighlightPrint("用户已被删除!\n");

break;

}

case TEACHING\_ASSISTANT:

{

tas = Remove(tas, del\_id);

HighlightPrint("用户已被删除!\n");

break;

}

default:

break;

}

UpdateFiles();

UpdateUsers();

return;

}

void Administrator::update()

{

admins = Remove(admins, id\_);

admins.push\_back(\*this);

UpdateFiles();

return;

}

std::ifstream &operator >>(std::ifstream &in, Administrator &admin)

{

in >> admin.identity\_;

in >> admin.id\_;

in >> admin.password\_;

return in;

}

std::ofstream &operator <<(std::ofstream &of, const Administrator &admin)

{

of << admin.identity\_ << '\n';

of << admin.id\_ << '\n';

of << admin.password\_ << '\n';

return of;

}

void Teacher::add\_course()

{

ClearScreen();

HighlightPrint("开设一门课程!\n");

std::cout << "课程ID: ";

std::string new\_id, new\_name;

int credit;

bool is\_optional, is\_scoring;

std::cin >> new\_id;

std::cin.get();

int index = Find(courses, new\_id);

if (index >= 0) // id符合

{

course\_id\_.push\_back(new\_id);

Course tmp = courses[index];

tmp.add\_teacher(id\_);

tmp.update();

update();

HighlightPrint("添加成功!\n");

MyGetCh();

return;

}

else

{

std::cout << "正在创建一门新课程...\n";

std::cout << "课程名称: ";

if (!(std::cin >> new\_name))

{

HighlightPrint("输入错误! \n");

MyGetCh();

return;

}

std::cin.get();

std::cout << "学分: ";

if (!(std::cin >> credit))

{

HighlightPrint("输入错误! \n");

MyGetCh();

return;

}

std::cin.get();

std::cout << "选修课? [0/1] ";

if (!(std::cin >> is\_optional))

{

HighlightPrint("输入错误! \n");

MyGetCh();

return;

}

std::cin.get();

std::cout << "是否记分? [0/1] ";

if (!(std::cin >> is\_scoring))

{

HighlightPrint("输入错误! \n");

MyGetCh();

return;

}

std::cin.get();

Course new\_course(new\_id, new\_name, credit, is\_optional, is\_scoring);

new\_course.add\_teacher(id\_);

course\_id\_.push\_back(new\_id);

update();

HighlightPrint("添加成功!\n");

MyGetCh();

return;

}

}

void Teacher::display\_class()

{

std::vector<Student> my\_class;

for (std::vector<Student>::iterator it = students.begin(); it != students.end(); it++)

{

if (it->class\_id() == class\_id\_)

my\_class.push\_back(\*it);

}

for (std::vector<TeachingAssistant>::iterator it = tas.begin(); it != tas.end(); it++)

{

if (it->Student::class\_id() == class\_id\_)

my\_class.push\_back(\*it);

}

std::sort(my\_class.begin(), my\_class.end(), CompareStudent);

ClearScreen();

HighlightPrint(class\_id\_ + '\n');

std::cout << "[按GPA排序]\n";

for (std::vector<Student>::iterator it = my\_class.begin(); it != my\_class.end(); it++)

it->display\_gpa\_info();

MyGetCh();

return;

}

void Teacher::update()

{

teachers = Remove(teachers, id\_);

teachers.push\_back(\*this);

UpdateFiles();

return;

}

std::ifstream &operator >>(std::ifstream &in, Teacher &t)

{

in >> t.identity\_;

in >> t.id\_;

in >> t.password\_;

in >> t.name\_;

char start\_flag;

while ((start\_flag = in.get()) == '\n')

;

if (start\_flag == '\*')

{

while (true)

{

char end\_flag;

while ((end\_flag = in.get()) == '\n')

;

if (end\_flag == '#')

break;

else

{

in.seekg(-1, std::ios::cur);

std::string tmp\_id;

in >> tmp\_id;

t.course\_id\_.push\_back(tmp\_id);

}

}

}

else

in.seekg(-1, std::ios::cur);

in >> t.is\_head\_teacher\_;

in >> t.class\_id\_;

return in;

}

std::ofstream &operator <<(std::ofstream &of, const Teacher &t)

{

of << t.identity\_ << '\n';

of << t.id\_ << '\n';

of << t.password\_ << '\n';

of << t.name\_ << '\n';

of << "\*\n";

for(std::vector<std::string>::const\_iterator it = t.course\_id\_.begin(); it != t.course\_id\_.end(); it++)

{

of << \*it << '\n';

}

of << "#\n";

of << t.is\_head\_teacher\_ << '\n';

of << t.class\_id\_ << "\n\n";

return of;

}

double Student::gpa()

{

int total = 0;

for (std::vector<Score>::iterator it = score\_.begin(); it != score\_.end(); it++)

total += it->num() \* courses[Find(courses, it->course\_id())].credit();

return (credit() ? (double(total) / double(credit())) : 0.0);

}

int Student::credit()

{

int total = 0;

for (std::vector<std::string>::iterator it = course\_id\_.begin(); it != course\_id\_.end(); it++)

total += courses[Find(courses, \*it)].credit();

return total;

}

void Student::add\_course()

{

ClearScreen();

HighlightPrint("课程ID: ");

std::string add\_id;

std::cin >> add\_id;

std::cin.get();

if (Find(courses, add\_id) < 0)

{

HighlightPrint("没有该课程! 请核对课程ID.\n");

MyGetCh();

return;

}

if (Find(course\_id\_, add\_id) >= 0)

{

HighlightPrint("已经添加过该课程!\n");

MyGetCh();

return;

}

course\_id\_.push\_back(add\_id);

update();

Course tmp = courses[Find(courses, add\_id)];

tmp.add\_student(id\_);

WriteCourses();

HighlightPrint("添加成功!\n");

MyGetCh();

return;

}

void Student::add\_score(Score score)

{

if (Find(score\_, score) < 0)

score\_.push\_back(score);

else

score\_[Find(score\_, score)].update\_num(score.num());

return;

}

void Student::course\_info()

{

ClearScreen();

for (std::vector<std::string>::iterator it = course\_id\_.begin(); it != course\_id\_.end(); it++)

{

Course tmp = courses[Find(courses, \*it)];

printf("%6s", tmp.id().c\_str());

printf("%24s", tmp.name().c\_str());

printf("%3d%s", tmp.credit(), "学分");

printf("%10s", tmp.is\_optional()?"选修课程":"必限课程");

if (tmp.is\_scoring())

{

if (Find(score\_, tmp.id()) < 0)

{

printf("%12s\n","成绩未录入");

continue;

}

Score score = score\_[Find(score\_, tmp.id())];

printf("%4d", score.num());

std::cout << " 课程内排名: " << score.rank() << '/' << tmp.student\_id().size() << std::endl;

}

else

printf("%12s\n","不记分");

}

std::cout << "总GPA: " << gpa() << std::endl;

MyGetCh();

return;

}

void Student::display\_gpa\_info()

{

std::cout << id\_ << ' ';

std::cout << name\_ << ' ';

std::cout << class\_id\_ << ' ';

std::cout << "GPA: " << gpa() << std::endl;

return;

}

void Student::update()

{

students = Remove(students, id\_);

students.push\_back(\*this);

UpdateFiles();

return;

}

std::ifstream &operator >>(std::ifstream &in, Student &stu)

{

in >> stu.identity\_;

in >> stu.id\_;

in >> stu.password\_;

in >> stu.name\_;

in >> stu.class\_id\_;

char start\_flag;

while ((start\_flag = in.get()) == '\n')

;

if (start\_flag == '\*')

{

while (true)

{

char end\_flag;

while ((end\_flag = in.get()) == '\n')

;

if (end\_flag == '#')

break;

else

{

in.seekg(-1, std::ios::cur);

std::string tmp\_id;

in >> tmp\_id;

stu.course\_id\_.push\_back(tmp\_id);

}

}

}

else

in.seekg(-1, std::ios::cur);

while ((start\_flag = in.get()) == '\n')

;

if (start\_flag == '\*')

{

while (true)

{

char end\_flag;

while ((end\_flag = in.get()) == '\n')

;

if (end\_flag == '#')

break;

else

{

in.seekg(-1, std::ios::cur);

Score tmp\_score;

in >> tmp\_score;

stu.score\_.push\_back(tmp\_score);

}

}

}

else

in.seekg(-1, std::ios::cur);

return in;

}

std::ofstream &operator <<(std::ofstream &of, const Student &stu)

{

of << stu.identity\_ << '\n';

of << stu.id\_ << '\n';

of << stu.password\_ << '\n';

of << stu.name\_ << '\n';

of << stu.class\_id\_ << '\n';

of << "\*\n";

std::vector<std::string>::const\_iterator it1;

for(it1 = stu.course\_id\_.begin(); it1 != stu.course\_id\_.end(); it1++)

{

of << \*it1 << '\n';

}

of << "#\n";

of << "\*\n";

std::vector<Score>::const\_iterator it2;

for(it2 = stu.score\_.begin(); it2 != stu.score\_.end(); it2++)

{

of << \*it2;

}

of << "#\n";

of << '\n';

return of;

}

bool CompareStudent(Student stu1, Student stu2)

{

return stu1.gpa() > stu2.gpa();

}

void TeachingAssistant::update()

{

tas = Remove(tas, id\_);

tas.push\_back(\*this);

UpdateFiles();

return;

}

std::ifstream &operator >>(std::ifstream &in, TeachingAssistant &ta)

{

in >> ta.identity\_;

in >> ta.id\_;

in >> ta.password\_;

in >> ta.Teacher::name\_;

in >> ta.Student::class\_id\_;

char start\_flag;

while ((start\_flag = in.get()) == '\n')

;

if (start\_flag == '\*')

{

while (true)

{

char end\_flag;

while ((end\_flag = in.get()) == '\n')

;

if (end\_flag == '#')

break;

else

{

in.seekg(-1, std::ios::cur);

std::string tmp\_id;

in >> tmp\_id;

ta.Teacher::course\_id\_.push\_back(tmp\_id);

}

}

}

else

in.seekg(-1, std::ios::cur);

while ((start\_flag = in.get()) == '\n')

;

if (start\_flag == '\*')

{

while (true)

{

char end\_flag;

while ((end\_flag = in.get()) == '\n')

;

if (end\_flag == '#')

break;

else

{

in.seekg(-1, std::ios::cur);

std::string tmp\_id;

in >> tmp\_id;

ta.Student::course\_id\_.push\_back(tmp\_id);

}

}

}

else

in.seekg(-1, std::ios::cur);

return in;

}

std::ofstream &operator <<(std::ofstream &of, const TeachingAssistant &ta)

{

of << ta.identity\_ << '\n';

of << ta.id\_ << '\n';

of << ta.password\_ << '\n';

of << ta.Teacher::name\_ << '\n';

of << ta.Student::class\_id\_ << '\n';

of << "\*\n";

std::vector<std::string>::const\_iterator it1;

for(it1 = (ta.Teacher::course\_id\_).begin(); it1 != (ta.Teacher::course\_id\_).end(); it1++)

{

of << \*it1 << '\n';

}

of << "#\n";

of << "\*\n";

std::vector<std::string>::const\_iterator it2;

for(it2 = (ta.Student::course\_id\_).begin(); it2 != (ta.Student::course\_id\_).end(); it2++)

{

of << \*it2 << '\n';

}

of << "#\n";

of << "\n";

return of;

}

**附录1：评分表**

课程名称： 计算机程序设计基础2

|  |  |  |
| --- | --- | --- |
| **项 目** | **评 价** | |
| 设计方案的合理性与创新性 | **3×2** |  |
| 设计与调试结果 | **4×2** |  |
| 设计说明书的质量 | **1×2** |  |
| 程序基本要求涵盖情况 | **4×2** |  |
| 程序代码编写素养情况 | **2×2** |  |
| 课程设计周表现情况 | **1×2** |  |
| 综合成绩 | **15×2** |  |

教师签名：

日 期：