面向对象程序设计与实践

(C++)2

学生成绩管理系统

程序说明书

班级：2014211319

姓名：刘含

学号：2014211684

班内序号：23

日期：2016/7/1

题目一

1. 一题题目以及需求分析

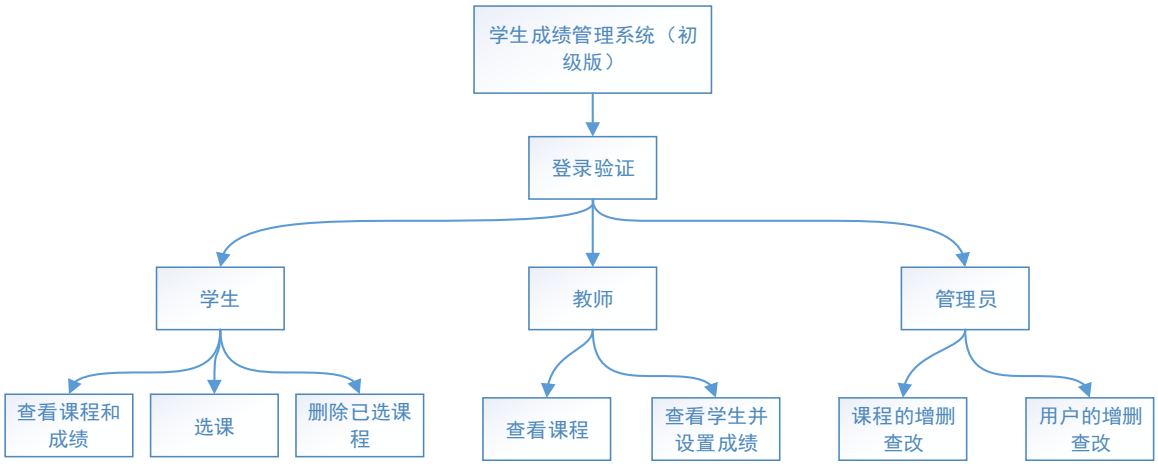
- 题目

题目一要求的是完成学生成绩管理系统初级版。其中，要求实现本地学生和教师两种用户的登录访问，并且，学生具有查看当前课程信息和选课、删除已选课程功能，教师具有查看任课课程和设置学生成绩功能。对于课程信息，要求包括课程编号、名称、学分，并在学生端显示学生成绩和针对性计算绩点，即必修课和选修课的绩点计算方式不同。

- 需求分析

1、 从宏观上对软件进行分析，考虑到软件的使用群体，决定选用图形化编程，提高软件的好感度。其中，C++的图形化库有 VC++.NET、QT 等等，这里选用了扩展性和移植性更为强大的 QT，便于开发多平台的软件。
2、 软件将实现开源，全部代码托管在 github 分布式版本控制系统中，为接下来三个版本的开发奠定了基础。
3、 对软件进行功能上的划分，主要分为用户和课程。考虑到真实性，对软件添加了管理员用户，可以进行课程和用户的增删查改功能，使得软件更加人性化。
4、 在用户层面，则可以分为学生、教师和管理员用户，每个用户具有一定权限的功能。学生包括：查看课程信息和对应成绩及绩点、选择选修课程、删除已选选修课。教师包括：查看任课信息、查看对应课程学生名单并设置成绩。管理员包括：查看系统所有必修课和选修课、任意添加、删除或者更新已有课程、查看系统所有用户、任意添加、删除或者更新已有用户。
5、 在课程层面，则可以分为必修课和选修课，这两个的不同主要体现在呈现方式和绩点计算方式的差异。

2. 各功能实现模块

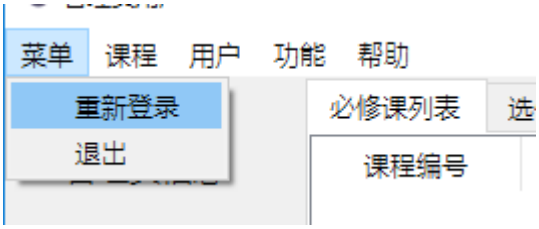


2.1 登录验证及注销功能

2.1.1 实现截图

1、登录界面

2、注销界面（菜单栏）



2.1.2 实现思路和代码

- ✧ 首先会创建一个继承自 QDialog 的界面，在完成系统初始化工作后首先呈现的便是该界面。
- ✧ 对于界面的验证登录功能，则会在创建界面时把自定义的一个系统环境（Envir）类的指针传递进去，登录界面可以调用该指针对应对象的接口进行验证。
- ✧ 在验证成功后，则会通过一个界面环境类（Envir_widget）的指针调取对应的用户界面，传递用户对象并显示。
- ✧ 重新登录时，只需要关闭当前界面并调用 Envir_widget 类相关接口呈现登录界面即可。

```
//寻找输入用户
Student* user1 = m_envir->checkUserStudent(username.toStdString(), md5_password.toStdString());
Teacher* user2 = m_envir->checkUserTeacher(username.toStdString(), md5_password.toStdString());
Admin* user3 = m_envir->checkUserAdmin(username.toStdString(), md5_password.toStdString());

//判断用户类型
if(user1){
    this->close();
    m_envir->setCourseStudent(user1);   //设置学生课程权限类
    m_envir_widget->showStudentWidget(user1);
}
else{
    if(user2){
        this->close();
        m_envir->setCourseTeacher(user2);   //设置教师课程权限类
        m_envir_widget->showTeacherWidget(user2);
    }
    else{
        if(user3){
            this->close();
            user3->activateEnvir(m_envir);      //设置管理员系统权限
            m_envir_widget->showAdminWidget(user3);
        }
        else{
            ui->wrong_label->show();
        }
```

2.1.3 原版、改进版

- ✧ 原版中登录用户密码均采用明文存储，十分不安全，在改进过程中，将所有密码均采用 MD5 的 32 位不对称加密方式，更为安全可靠。代码如下：

```
//密码采用md5加密
QByteArray temp = QCryptographicHash::hash(password.toLatin1(), QCryptographicHash::Md5);
QString md5_password = temp.toHex();
```

2.1.4 用到的重点知识简介

图形化界面当然需要用到 QT 的相关知识，包括界面的创建，控件的添加以及信号与槽函数的链接。

2.1.5 心得

在实现登录界面的过程中，最关键的就是验证和界面互调，其中，用户验证通过设计系统环境类(Envir)的接口解决；界面互调早期都是通过界面关闭状态传递到 Main 函数判断，这种方法只能实现两种状态的判断，限制比较多，在界面比较多时便难以实现。开发过程中，这一问题通过类似系统环境类(Envir)的方法解决，设置一个界面环境(Envir_widet)类，包含了所有的界面指针和界面相关接口，这样每一个界面都可以去直接调用该接口来实现界面互调，十分方便和可靠。

2.2 系统环境功能

2.2.1 功能介绍

作为与数据库相同的功能，包括系统和界面两部分，系统类（Envir）包括了所有的用户和课程信息，界面类（Envir_widget）包括了所有的界面指针信息。

2.2.2 实现的思路和代码

- ✧ Envir 类内通过两个标准容器类 set 存储必修课和选修课类指针
- ✧ Envir 类内通过三个标准容器类 map 存储学生、教师和管理员信息，key 为类指针，value 为 md5 密码
- ✧ 对于增删查改功能均实现了相关的接口

Envir:

```
    void setCourseStudent(Student* student);
    void setCourseTeacher(Teacher* teacher);

    void addUserStudent(Student* student, std::string password);
    void addUserTeacher(Teacher* teacher, std::string password);
    void addUserAdmin(Admin* admin, std::string password);

    std::map<Student*, std::string> getUserStudent();
    std::map<Teacher*, std::string> getUserTeacher();
    std::map<Admin*, std::string> getUserAdmin();

    User* findUser(std::string id);
    void changeUserPass(User* user, std::string pass);

    void deleteUser(User* user);

    Student* checkUserStudent(std::string username, std::string password);
    Teacher* checkUserTeacher(std::string username, std::string password);
    Admin* checkUserAdmin(std::string username, std::string password);

private:
    //保存当前系统课程
    std::set<Course*> m_obligatory_course;
    std::set<Course*> m_elective_course;

    //保存用户登录信息(md5密码)
    std::map<Student*, std::string> m_student;
    std::map<Teacher*, std::string> m_teacher;
    std::map<Admin*, std::string> m_admin;
};
```

Envir_widget:

```
//界面环境
class Envir_widget
{
public:
    void setWidget(Login* login, MainWindow_student* student, MainWir

    //打开指定界面
    void showLoginWidget();
    void showAdminWidget(Admin* user);
    void showStudentWidget(Student* user);
    void showTeacherWidget(Teacher* user);

private:
    //保存当前界面指针
    Login* m_widget_login;
    MainWindow_admin* m_widget_admin;
    MainWindow_student* m_widget_student;
    MainWindow_teacher* m_widget_teacher;
};
```

2.2.3 原版、改进版

其中 Envir 类在初始架构中就已经考虑到了，但对于 Envir_widget 类之前是想采用界面直接调用的方式进行，后面开发中才改进为通过设置环境中间层进行接口调用的方式，改进后系统扩展性得到了进一步的加强。

2.2.4 用到的重点知识简介

标准容器类等。

2.2.5 心得

这两个类属于关键的系统架构设计部分，应该说是早期开发的任务，做的过程中可以说是逐步完善的，其他模块需要什么样的接口就在这两个类中添加对应的函数。编程中对于容器类的使用可以说是得到了十分大的训练，其中，在 map 和 set 容器中，由于其底层的数据结构限制，要求 key 必须重载小于号，这里采用的是指针存储方式。开始时直接在对应类中重载，发现并没有什么作用，原来指针直接作为整数比较大小，便不了了之了。

2.3 学生和教师查看自己课程信息

2.3.1 实现截图

1、学生

2、教师

2.3.2 实现的思路和代码

- ✧ 对于课程设置一个基类 Course,派生为必修课 Obligatory_course 和选修课 Elective_course，这几个类均具有最高权限。
- ✧ 由于是本地版，如何能够准确地获取到对应用户对课程操作的权限十分重要，因此对于最高权限类课程做了进一步封装为 Course_user，再派生学生为 Course_student 类，教师为 Couse_teacher 类，基类 Course_user 中包含一个保护变量指针 Course*，这样便可以通过委派调用的方式对权限进行划分。
- ✧ 设置用户类 User 基类，派生出 Student,Teacher 和 Admin 三种用户类型，在 Student 和 Teacher 类内包含一个私有变量，为一个包含了该用户所属课程的 set 容器，类型为 Course_student*或者 Course_teacher*，即是封装权限后的课程类。
- ✧ 每次进入界面便传递对应用户指针进去，并初始化该用户的相关权限课程，这里需要注意学生和教师的所属课程均是在登录时候动态创建的，系统中唯一的可区分对象只有最高权限课程和用户。

1、最高权限课程

```
//必修课
class Obligatory_course: public Course
{
public:
    using Course::Course;
    using Course::operator<;

    float calculateGPA(Student* student) final override;    //重载课程中计算GPA的虚函数

    int getCourseType() final override;
};
//选修课
class Elective_course: public Course
{
public:
    using Course::Course;
    using Course::operator<;

    float calculateGPA(Student* student) final override;    //重载课程中计算GPA的虚函数

    int getCourseType() final override;
};
```

2、包装权限后的课程

```
//重新包装权限后的课程类
class Course_user
{
public:
    Course_user(){}
    Course_user(Course* course){
        this->m_course = course;
    }
    ~Course_user(){}

    bool operator < (const Course_user* course)
    {
        return m_course->getID() < course->m_course->getID();
    }

    //学生与教师公共权限
    std::string getID();
    std::string getName();
    int getCredit();
    int getCourseType();
    int getCapicity();

protected:
    Course* m_course;
};
```

3、用户所属课程

```
    std::set<Course_student*> m_course;              //学生对应课程
    std::set<Course_student*> m_select_course;       //学生对应可选课程


private:
    std::set<Course_teacher*> m_course;      //教师任课课程
};
```

2.3.3 原版、改进版

在开发过程中，考虑到权限的控制，对于基类 Course 进行了改进，进一步对不同权限进行封装为不同用户课程类，更为安全可靠。

2.3.4 用到的重点知识简介

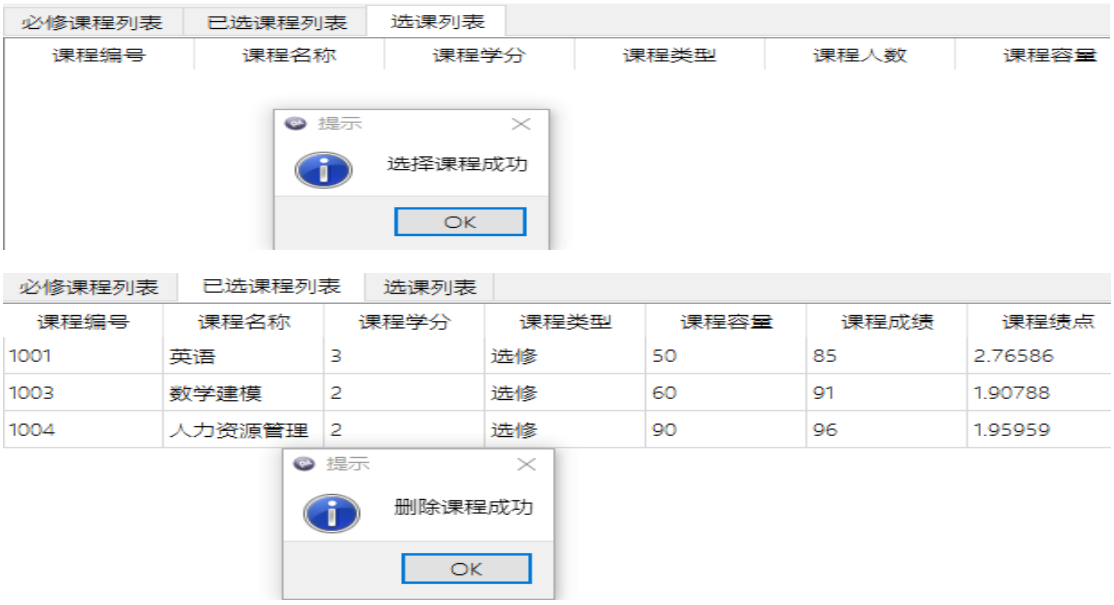
继承与派生，保护变量，委派构造函数，运算符重载，虚函数

2.3.5 心得

这个功能中对于权限的封装可以说是程序中一个亮点，对于一个全新十分大的类，在发布中就会出现各种安全性问题，如何能够将这些权限针对指定用户进行封装，限制权限才能更好地利于类的发布和共享使用。其中，面向对象编程的个思想便是封装，在这里得到了一定的训练和体验。

2.4 学生选课和删除课程

2.4.1 实现截图



2.4.2 实现的思路及代码

- ✧ 选课其实是 Course_student 类的一个功能，学生可以通过获取可选课程，其中每一个课程类型都是封装后的 Course_student，进一步调用该类的方法把学生自己添加到对应课程中，再由 Course_student 进一步更新系统环境。
- ✧ 删除课程与选课十分类似，只是把添加学生改为删除学生。

1、选课

```
m_user->addCourse(temp_find);
updateTable();      //更新用户列表信息
QMessageBox::information(this, QString::fromLocal8Bit("提示"), QString::fromLocal8Bit("选择课程成
```

```
void Course_student::addElectiveStudent(Student* student
{
    //判断待增加课程是否为选修课
    if (!this->getCourseType()) {
        m_course->addStudent(student);
    }
    else{
        throw AuthorityError();
    }
}
```

2、删除课程

```
bool Course_student::deleteElectiveStudent(Student* student)
{
    if (!this->getCourseType()) {
        return m_course->deleteStudent(student);
    }
    return false;
}
```

2.4.3 原版、改进版

暂无

2.4.4 用到的重点知识简介

异常处理，QT 列表处理

2.4.5 心得

在前面对于架构的设计基础上，增加这一个功能显得十分简单，相关接口早已设计好。这样来说，一个好的架构设计还是十分重要的。

2.5 教师设置学生成绩

2.5.1 实现截图

2.5.2 实现的思路及代码

- ✧ 设置成绩的第一步就是获取对应课程的学生信息，这个功能由封装权限后的 Course_teacher 类提供，直接获取学生对象指针 Student*
- ✧ 设置成绩的时候，先查找对应 ID 的学生对象，再把这个对象和成绩传给 Course_teacher 对应的设置成绩函数即可，而实际最终的修改都是在最高权限类 Course 中进行的。

```
void Course::setGrade(std::pair<Student*, float> student_grade)
{
    auto i = m_student.find(student_grade.first);
    if (i != m_student.end()) {
        if(student_grade.second >= 0){
            i->second = student_grade.second;
        }
        else{
            throw std::invalid_argument(QString::fromLocal8Bit("学生成绩不能为负数").toStdString());
        }
    }
}
```

2.5.3 原版、改进版

改进版中加入了异常处理机制，防止教师设置学生成绩为负数造成系统崩溃

2.5.4 用到的重点知识

异常处理

2.5.5 心得

教师设置学生成绩这个功能都是在前期做基础类时预留好的，在后期一步一步实现具体功能时只需要调用接口就可以实现，这里没有什么难点。

2.6 管理员

2.6.1 实现截图

2.6.2 实现的思路及代码

- ✧ 管理员这个功能是为了方便管理者对课程或用户增删查改，其中这些功能都必须基于最高权限进行，因此，对于管理员我们没有像学生和教师那样封装权限类，而是直接把系统环境（Envir）传递进去。

✧ 增删查改功能直接通过修改系统环境中的容器类内内容实现。
✧ 在界面上为了方便操作，增加了用户详细信息或者课程详细信息界面。

1、添加用户

```
void Envir::addUserStudent(Student* student, std::string password)
{
    m_student.insert(std::make_pair(student, password));
}

void Envir::addUserTeacher(Teacher* teacher, std::string password)
{
    m_teacher.insert(std::make_pair(teacher, password));
}

void Envir::addUserAdmin(Admin* admin, std::string password)
{
    m_admin.insert(std::make_pair(admin, password));
}
```

2、为课程添加用户

```
void Information_course::addStudent(Course * course)
{
    course->clearStudent();
    int row = ui_student_model_y->rowCount();
    if(row != -1){
        for(int i = 0; i < row; i++)
        {
            QString id = ui_student_model_y->item(i, 0)->text();
            User* user = m_user->getEnvir()->findUser(id.toStdString());
            if(user->getUserType() == user_type::student){
                course->addStudent((Student*)user);
            }
        }
    }
}
```

2.6.3 原版、改进版

早期未实现对课程动态添加学生和为学生动态添加课程，后期做了界面后实现了这一点，进一步增强了管理员的功能，系统更加灵活。

2.6.4 用到的重点知识简介

QT 界面的设计，异常处理，信号与槽机制

2.6.5 心得

管理员是这个软件的第二个亮点，这个用户集结了系统的最高权限，因此在实现上遇到了很多困难，相信对自己的 debug 功能得到了很大的训练。这也是最后一个实现的部分，代码量一度达到 3000 行左右，经过不断地重构和优化代码逻辑，程序看起来更加流畅、易懂，调试起来也十分方便。

题目二

1. 二题题目以及需求分析

- 题目

  这一版要求的是学生成绩管理系统高级版，即在第一版的基础上增加了文件存取和泛型排序功能。

- 需求分析

  1、 对于文件存取可以直接采用文本文件存储，比较方便，只需要定义好读写格式就行，文件内应该包含了 Envir 类的全部信息。

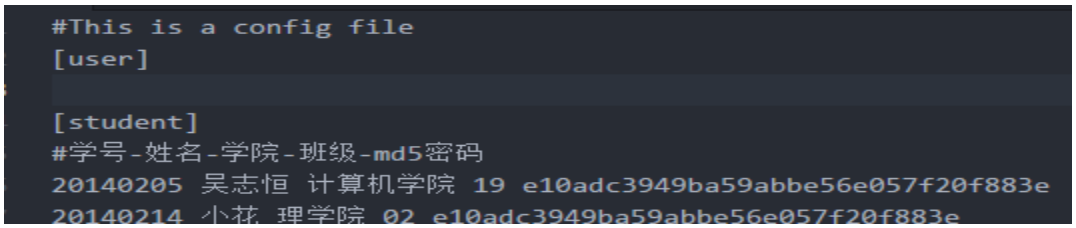
  2、 泛型排序可以实现学生按课程 ID 排、成绩排、绩点排，或者教师按照学生 ID 排、成绩排，同时还可以实现逆向排序。

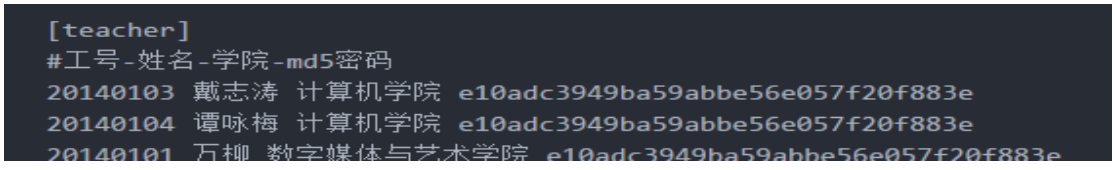
2. 各功能实现模块

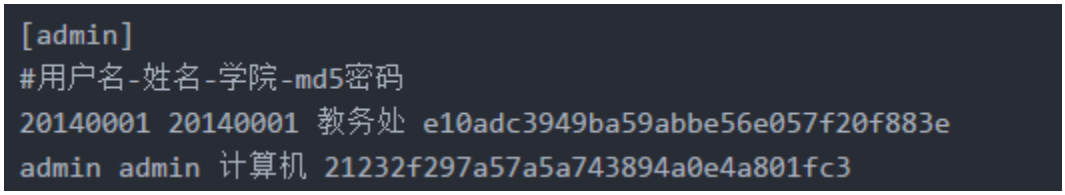
题目二要求的功能有文件存取、泛型排序，现将各功能实现细明叙述如下。

2.1 文件存取功能

2.1.1 实现截图







```
[obligatory]
#课程编号-课程名称-学分-课程容量-教师工号-学生及成绩
0003 概率论 4 50 20140106 <20140203, 92> <20140201, 94> <20140202, 95> <20140207, 91>
0004 计算机组成原理 5 60 20140103 <20140205, 88> <20140203, 30> <20140201, 91> <20140202, 91> <20140204, 68>
0006 计算机网络 4 60 -1 <20140201, 92>
```

```
[elective]
#课程编号-课程名称-学分-课程容量-教师工号-学生及成绩
1002 红楼梦 2 50 20140101 <20140214, -1>
1004 人力资源管理 2 90 20140102 <20140214, -1> <20140201, 96>
```

2.1.2 实现思路和代码

- ✧ 增加了一个模块 Config，实现对配置文件的读和写
- ✧ 其中读取配置文件主要通过一个自动机的状态转移实现对不同课程或者用户的读取创建，具体细节匹配还是通过正则表达式实现的
- ✧ 写配置文件内部直接把 Envir 内容一次写入配置文件，这个函数在每次修改 Envir 时被触发

1、 读配置文件

```
//定义正则表达式匹配配置文件
std::regex pattern("\\[(.*)\\]");
std::string str;
std::match_results<std::string::const_iterator> result;

//一行一行读取
config_state state = config_state::nop;
while(!file.eof()){
    std::getline(file, str);
    if(str[0] == '#' || str == "")        //一行为注释或者为空
        continue;

    //确定下一个状态
    if(std::regex_match(str, result, pattern)){ ...}

    std::stringstream file_line(str);
    switch (state) {
    case config_state::student: {...}

    case config_state::teacher: {...}

    case config_state::admin: {...}

    case config_state::obligatory: {...}

    case config_state::elective: {...}

    default:
        break;
    }
```

2、 写配置文件

```
std::ofstream file;
file.open("config.ini", std::ios_base::out);
file << "#This is a config file\n" << "[user]\n";

file << "\n[student]\n" << "#学号-姓名-学院-班级-md5密码\n";
for(auto i : this->m_envir->getUserStudent())
{
    file << *(i.first) << " " << i.second << "\n";
}

file << "\n[teacher]\n" << "#工号-姓名-学院-md5密码\n";
for(auto i : this->m_envir->getUserTeacher())
{
    file << *(i.first) << " " << i.second << "\n";
}

file << "\n[admin]\n" << "#用户名-姓名-学院-md5密码\n";
for(auto i : this->m_envir->getUserAdmin())
{
    file << *(i.first) << " " << i.second << "\n";
```

2.1.3 原版、改进版

暂无

2.1.4 用到的重点知识简介

自动机状态转移、正则表达式、文件读写、重载运算符

2.1.5 心得

这个应该是第二版的主要功能了，实现的代码最多，其中读配置文件采用了专业课自动机中的状态转移实现，觉得还是比较理想的，而细节的匹配用到了 C++11 的 regex 正则表达式模块。另外由于采用中文字符，对于编码格式显得尤为重要，第一版中由于在 windows 下采用了 gbk 编码，在写入文件或者读取时常常会产生乱码现象，后面便统一到了 UTF-8 编码，完美解决。我想这一个功能含金量还是挺高的，在不断地学习中创造代码。

2.2 泛型排序功能

2.2.1 实现截图

课程编号	课程名称	课程学分	课程类型	课程容量	课程成绩	课程绩点
1004	人力资源管理	2	选修	90	96	1.95959
1003	数学建模	2	选修	60	91	1.90788
1001	英语	3	选修	50	85	2.76586

课程编号	课程名称	课程学分	课程类型	课程容量	课程成绩	课程绩点
1003	数学建模	2	选修	60	91	1.90788
1004	人力资源管理	2	选修	90	96	1.95959
1001	英语	3	选修	50	85	2.76586

2.2.2 实现的思路和代码

- ✧ 泛型排序应用到了泛型编程的思想，通过把一个模板类的 Vector 传到排序函数内，指定排序方向，便可以实现
- ✧ 其中具体的排序函数应用的是基础的冒泡排序
- ✧ 调用函数时给模板传的是封装后结构体

```
//泛型排序算法
namespace Sort {
    template<typename T>

    //mode可以选择按某一项进行排序，reverse可以选择是否反向排序
    void sortVectorCourse(std::vector<T>& course, std::string mode, bool reverse = false)
    {
        bool flag = true;
        for(int i = 0; i < course.size(); i++)
        {
            flag = true;
            for(int j = i; j < course.size(); j++)
            {
                if(mode == "id"){ ...}
                else if(mode == "grade"){ ...}
                else if(mode == "gpa"){ ...}
            }
            if(flag){
//                return;
            }
        }
    }
}
```

2.2.3 原版、改进版

无

2.2.4 用到的重要知识简介

泛型编程、模板类、冒泡排序

2.2.5 心得

对于泛型编程，自己之前比不太熟悉，编程中也是遇到了很多问题，在考虑了很久后，决定将表格内容重新构造为结构体，对结构体泛型排序，虽然没有那么完美，但也是实现了应有的功能。

题目三

1. 三题题目以及需求分析

- 题目

  这一版要求的是网络版，能够多用户同时登陆以及数据同步。

- 需求分析

  1、 首先是选用客户端与服务器通信方式，一般包括 TCP 和 UDP，这里选用

了面向连接的 TCP，数据传输更加稳定可靠。

2、 对于多用户登陆，这里在服务器端对于每一个套接字分配一个独立线程，服务器完全下放权限到线程，由每一个套接字的线程独立完成对应客户端的请求。

3、 考虑到可移植性和可扩展性，选用了 QT 的 tcp 模块编程，并把服务端做成命令行方式，方便直接部署到云主机上。

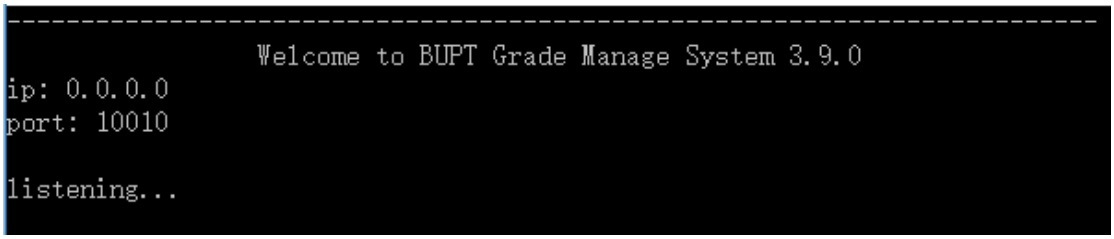
4、 客户端与服务端通信格式选用了更为方便的 JSON 文本传输格式

2. 各功能实现模块

题目一要求的功能有网络同步，多用户登录，先将各功能实现细明叙述如下。

2.1 网络功能（网络访问、多用户、同步）

2.1.1 实现截图



2.1.2 实现思路和代码

✧ 在服务端，创建一个继承自 QTcpserver 的 Conver_server 类，以及继承自 QThread 的 Convey_thread 类，当 Convey_server 监听到一个新的连接时，便开启一个新 Convey_thread 线程

✧ 在线程内，阻塞直至客户端发送请求，对于任意请求均按照 JSON 解析处理并回发相应信息

✧ 其中请求包括三种：”verify”, “get”, “post”，分别代表客户端请求验证用户、请求传送信息、请求修改信息

1、服务端解析

```
    QJsonValue jsonvalue;
    if(jsondoc.isObject())
    {
        jsonobj = jsondoc.object();

        //客户端请求验证
        if(jsonobj.contains("verify"))
        {
            jsonvalue = jsonobj.take("verify");
            if(jsonvalue.isObject())
            {
                return translateVerify(jsonvalue.toObject());
            }
        }

        //客户端请求数据
        if(jsonobj.contains("get"))
        {
            jsonvalue = jsonobj.take("get");
            return translateGet(jsonvalue);
        }

        //客户端请求修改数据
        if(jsonobj.contains("post"))
        {
            jsonvalue = jsonobj.take("post");
            if(jsonvalue.isObject())
            {
                return translatePost(jsonvalue.toObject());
            }
        }
    }
}
```

3、 服务端处理”verify”

```
QByteArray Convey_thread::translateVerify(QJsonObject jsonobj)
{
    QJsonValue jsonvalue;
    QJsonObject tempobj;

    QString username, password;
    if(jsonobj.contains("username")) {...}
    if(jsonobj.contains("password")) {...}

    User* temp_user = m_envir->checkUserStudent(username.toStdString(), password.
    if(!temp_user){ ...}

    //验证用户成功
    if(temp_user){ ...}
    else{
        tempobj.insert("ack", false);

        m_user = NULL;
    }

    QJsonDocument jsondoc;
    jsondoc.setObject(tempobj);
    return jsondoc.toJson();
}
```

4、 服务端处理”GET”

```
    if(get_value.isString())
    {
        QString get_string = get_value.toString();

        //获取当前课程
        if(get_string == "cur_course") {...}

        //学生获取可选课程
        else if(get_string == "sel_course") {...}

        //管理员获取所有用户
        else if(get_string == "user") {...}
    }

    else if(get_value.isObject())
    {
        QJsonObject get_object = get_value.toObject();

        //获取课程详细信息
        if(get_object.contains("course_info")) {...}

        //获取用户详细信息
        if(get_object.contains("user_info")) {...}
    }

    jsondoc.setObject(jsonobj);
    return jsondoc.toJson(QJsonDocument::Compact);
```

5、 服务端处理”POST”

```
    //学生选择选修课
    if(jsonobj.contains("sel_course")) {...}

    //学生删除选修课或者管理员删除课程
    if(jsonobj.contains("del_course")) {...}

    //获取当前课程id
    QString course_id = "";
    if(jsonobj.contains("course_id")) {...}

    //教师更改学生成绩
    if(jsonobj.contains("stu_grade")) {...}

    //管理员删除用户
    if(jsonobj.contains("del_user")) {...}

    //管理员添加课程
    if(jsonobj.contains("add_course")) {...}

    //管理员更新课程
    if(jsonobj.contains("update_course")) {...}

    //管理员添加用户
    if(jsonobj.contains("add_user")) {...}

    //管理员更新用户
    if(jsonobj.contains("update_user")) {...}

    jsondoc.setObject(rtnobj);
    return jsondoc.toJson(QJsonDocument::Compact);
```

2.1.3 原版、改进版

这一版对于登录界面做了优化和改进，软件更加人性化，友好度得到了提升。

2.1.4 用到的重点知识简介

JSON，QTcpser，QTcpsocket，线程。

2.1.5 心得

第三版则更为接近真实情况，网络版首先就是要对第二版的代码进行分离，将服务端代码和客户端部分分离为两个工程，重新修改相关程序逻辑，这一步比较麻烦，花了一天左右时间才弄完。在客户端加上管理员用户则是最为复杂的，主要表现在与服务器端交互进行课程的增删查改，涉及的格式比较多，服务端解析 JSON 代码比较复杂，这一版代码量也是达到了 6000 余行，是第一版的二倍之多。

其中遇到的一个主要问题是 TCP 的大容量字节传送问题，其中发送端会分批发送大字节，必须在发送完后阻塞住等待发送完再继续监听，而接收端受到缓存区容量限制，一次只能读取一定字节的字节流，必须在读取完后等待 1ms 判断是否还有字节，不断进行循环判断，直至读取完成。这一块开始没有注意，在传送大字节时便会发送 JSON 无法解析，导致客户端无法正常显示。