



东华理工大学
EAST CHINA INSTITUTE OF TECHNOLOGY

课程设计报告

C++面向对象程序设计

职工档案管理系统

学 号: 2022212543

学生姓名: 夏振华

专 业: 通信工程

班 级: 2221302

指导教师: 涂其远

2024 年 5 月 30 日

目录

一、课程设计目的和要求	1
二、需求分析	1
三、概要设计	1
1. 用户模块	2
2. 职工档案模块	2
四、详细设计和思路	3
1. 中英双语言功能	3
2. 背景音乐和语音播报	3
3. 注册	4
4. 登录	5
5. 忘记密码	7
6. 查询职工档案信息	8
7. 新增职工档案信息	9
8. 修改职工档案信息	10
9. 删除职工档案信息	15
五、运行环境及使用说明	15
六、项目总结与展望	16
参考文献	17
附录	17
1. Main.cpp	17
2. User.h	34
3. User.cpp	35
4. EmployeeProfile.h	42
5. EmployeeProfile.cpp	44
6. Language.h	57
7. Chinese.h	59
8. Chinese.cpp	60
9. English.h	63
10. English.cpp	64
11. Constant.h	68
12. MD5.h	69

一、课程设计目的和要求

C++面向对象课程设计是通信工程专业的集中实践性环节之一，是学习完《C++面向对象程序设计》课程后进行的一次全面的综合练习。通过 C++面向对象程序设计课程设计，让学生能全面理解、掌握面向对象的基本知识和技能，培养学生利用面向对象程序设计方法分析问题、解决问题的能力；培养学生针对具体的应用和实际问题，综合运用所学知识对问题进行抽象，分析、设计的能力，使学生掌握面向对象程序的编程方法。通过 C++面向对象课程设计教学，培养学生严谨求实的科学态度，激发学生的求知热情、探索精神、创新欲望，提高学生的综合素养。通过 C++面向对象程序设计课程设计，让学生熟悉面向对象基本理论和知识；掌握面向对象程序设计方法；初步掌握利用面向对象程序设计方法解决实际问题的技能。

设计职工档案管理系统，实现职工档案信息的显示、录入、删除、修改、排序、查询（可以按多种方式查询）等功能，数据存储于文件中。

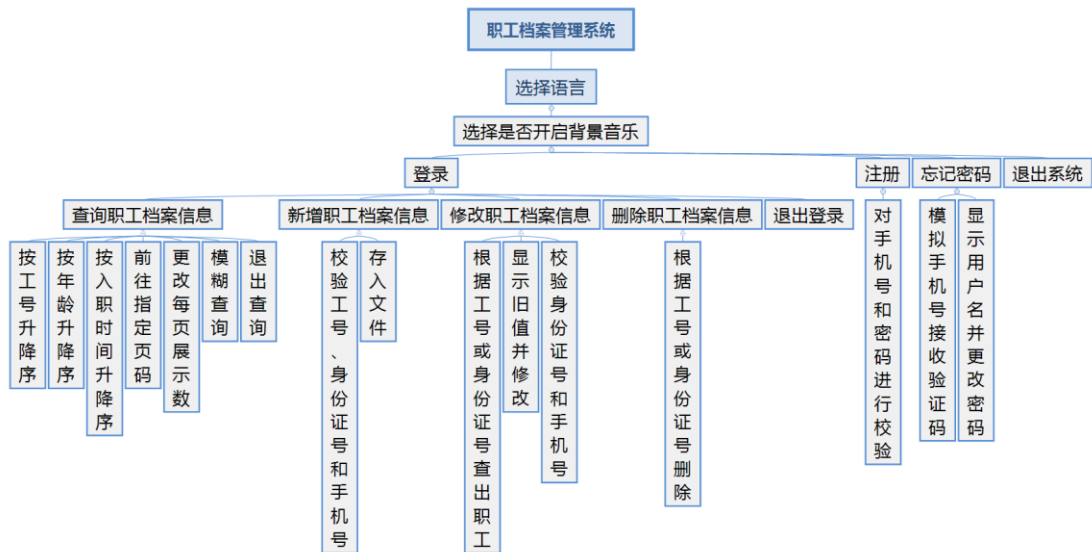
二、需求分析

本课程设计选题为“职工档案管理系统”。系统主要功能是管理职工档案信息，要求能实现但不限于：

1. 显示职工档案信息
2. 录入职工档案信息
3. 修改职工档案信息
4. 删除职工档案信息
5. 对员工工号、入职日期等进行排序
6. 通过工号、身份证查询职工档案信息
7. 模糊查询职工档案信息
8. 登录、注册
9. 忘记密码

三、概要设计

针对系统的设计要求，确定本系统的结构如下图：



此设计主要分了两个层面，一个是用户模块，一个是职工档案模块。

1. 用户模块

该模块可以为用户提供登录、注册和修改密码的功能。用户可以通过输入用户名和密码来进行登录，如果尚未拥有账户，则可以进行注册。同时，用户还可以在忘记密码时通过手机号进行修改密码。所有用户数据会被写入到名为“user.txt”的文件中，以便进行持久化存储。

2. 职工档案模块

该模块提供了对职工档案信息进行增加、删除、修改和查询的功能。用户可以根据需要对职工档案信息进行各种操作，包括对已有信息进行修改和删除不再需要的信息，或者新增新的档案信息。

在进行查询操作时，用户可以选择进行排序查询，按照指定的字段对档案信息进行排序，以便更加方便地浏览和管理数据。此外，模糊查询功能也是这个模块的一大特色，用户可以根据部分关键词来查询符合条件的档案信息，从而快速定位到需要的数据。

所有职工档案数据会被写入到名为“employee.txt”的文件中，以便进行持久化存储。这样不仅可以保证数据的安全性和可靠性，还可以方便后续的数据管理和分析工作。

四、详细设计和思路

1. 中英双语言功能

此功能鉴于本课程设计缺少亮点，思来想去决定实现一下双语言系统。此功能利用了诸多 C++ 课程知识，比如：继承、静态成员变量、纯虚函数和函数重写等等。除此之外，还使用了智能指针等。

此功能一开始是想通过简单的对变量名拼接实现，如：

```
string EnString = "_ENGLISH";  
string resultString = "";  
const string WELCOME_LOGIN = "欢迎登录职工档案管理系统";  
const string WELCOME_LOGIN_ENGLISH = "Welcome to the Employee  
Profile Management System";
```

对用户选择进行判断，当为英文时 resultString 被赋值为 "_ENGLISH" 得到最终的输出语言：

```
cout << WELCOME_LOGIN + resultString;
```

理想输出为：Welcome to the Employee Profile Management System

实际输出为：欢迎登录职工档案管理系统_ENGLISH

但是发现，将一个变量名和一个变量的值进行拼接得到一个新的变量名，再输出这个新变量的值的实现有点困难，查阅资料后发现可以改为工厂模式实现双语言功能。

在 main 函数中定义一个工厂函数和一个智能指针类型的全局变量，根据用户选择的语言创建不同的对象(自定义类 Chinese 或 English)，赋值给全局变量，然后利用这个全局变量调用类中的函数或获取类中的属性。并且 Chinese 类和 English 类继承于同一个基类 Language，便于统一函数和成员变量。

2. 背景音乐和语音播报

背景音乐通过 mciSendString 实现，用于播放 “.mp3” 格式的文件。背景音乐有五首音乐，如图所示：

名称	#	标题	参与创作的艺术家	唱片集
1.mp3		瞬间的永恒(钢琴曲)	赵海洋	夜色钢琴曲
2.mp3		いつも何度でも	宗次郎	Prime Selection
3.mp3		夜空中最亮的星(钢琴版)	赵海洋	
4.mp3		Eternity	David Foster	Eleven Words
5.mp3		Between Worlds	Roger Subirana	X II

在自定义函数中,通过设置随机数种子和 `rand()` 函数生成一个随机的数字,拼接到最终选择的文件名中。

语音播报通过 `PlaySound` 实现,用于播放 “.wav” 格式的文件。语音播报功能整合了一个函数,函数中会对传过来的文件名参数进行拼接,选择相应的语言和对应的语音进行播报。

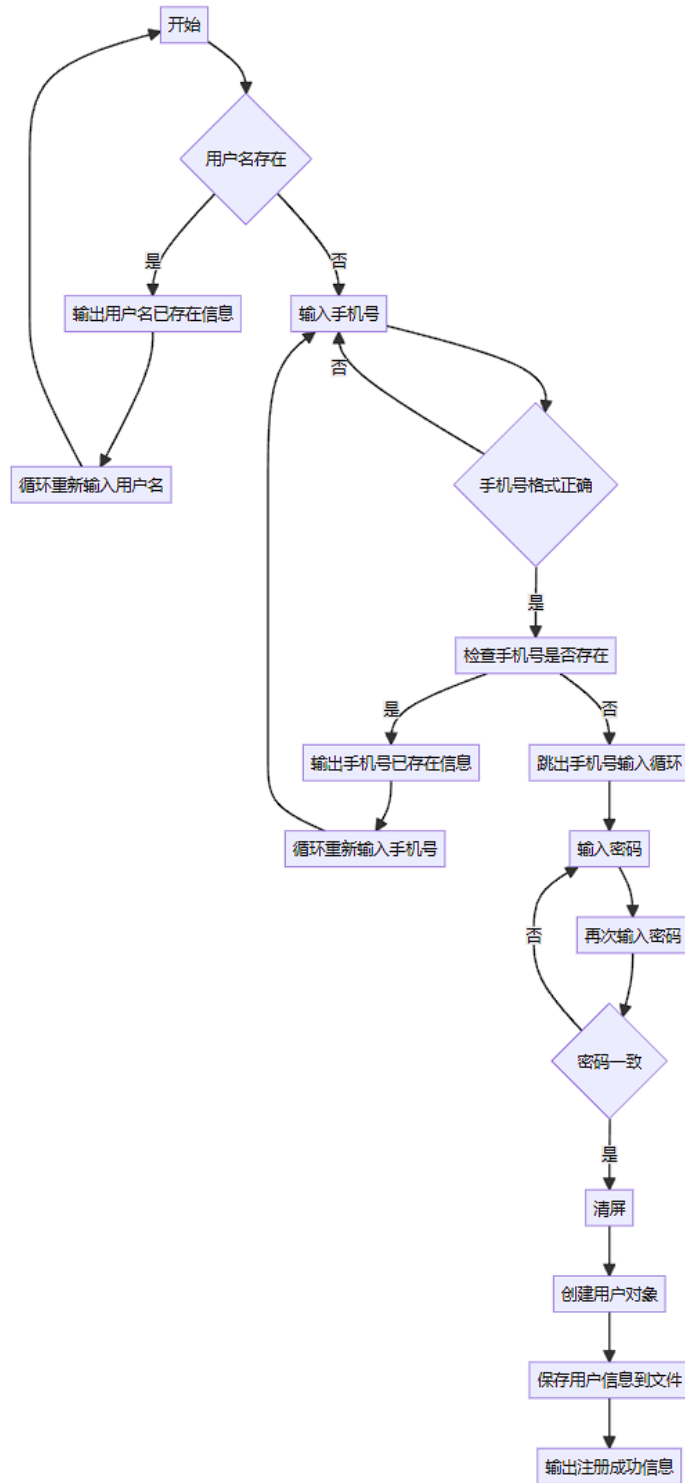
3. 注册

用户注册时,会通过自定义函数对用户名和手机号是否存在进行校验,并且也通过正则表达式校验手机号的格式。

在用户输入密码时,调用用户输入密码的函数,会使密码显示为 “*”,保护了用户的隐私,同时也会校验密码的长度,还会要求用户再次输入密码,以防输错。并且在存储密码时,也会使用 `MD5` 加密算法对密码进行加密。

```
欢迎注册职工档案管理系统
请输入用户名: admin
用户名已存在, 请重新输入
请输入用户名: admin1
请输入手机号: 18000001111
手机号已存在, 请重新输入
请输入手机号: 12345678910
手机号格式错误!
请输入手机号: 18011112222
请输入密码: ***
密码长度不能少于6位, 请重新输入: *****
密码长度不能超过20位, 请重新输入: *****
请再次输入密码: *****
两次密码不一致, 请重新输入!
请输入密码: *****
请再次输入密码: *****|
```

流程图:



4. 登录

输入用户名和密码进行登录。

当用户名不存在时会退出登录功能，效果如图：

```

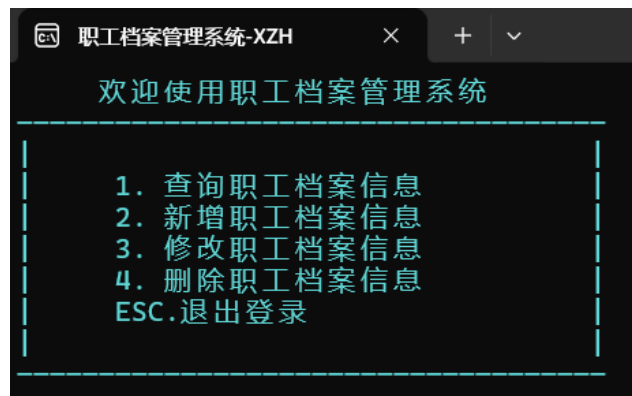
    欢迎登录职工档案管理系统
    请输入用户名: admin2
    用户名不存在, 请先注册
    2024/05/30 周四
    16:52
    欢迎使用职工档案管理系统
    *****
    *****
    *****      1. 登录      *****
    *****      *****
    *****      2. 注册      *****
    *****      *****
    *****      3. 忘记密码  *****
    *****      *****
    *****      ESC.退出系统 *****
    *****      *****
    *****
    *****
    
```

用户名和密码不匹配时会重新输入，效果如图：

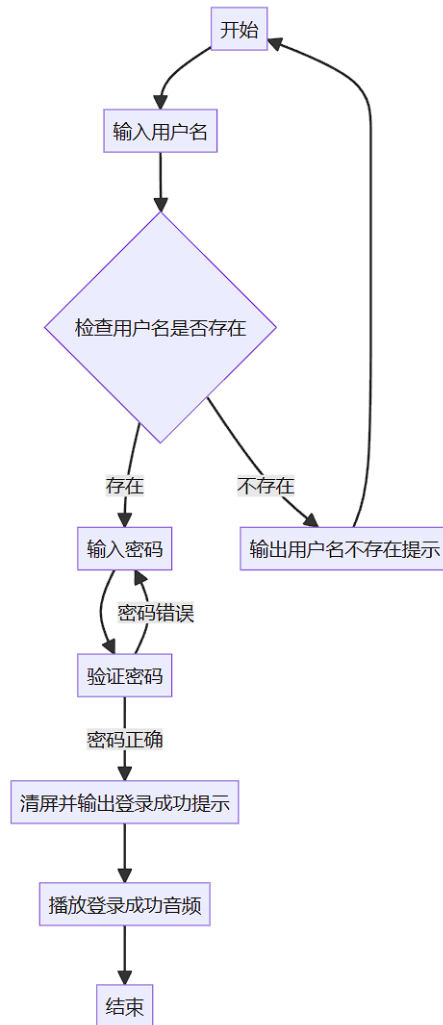
```

    欢迎登录职工档案管理系统
    请输入用户名: admin
    请输入密码: ***
    密码长度不能少于6位, 请重新输入: *****
    用户名或密码错误, 请重试!
    请输入用户名:
    
```

登录成功后，进入主界面



流程图：

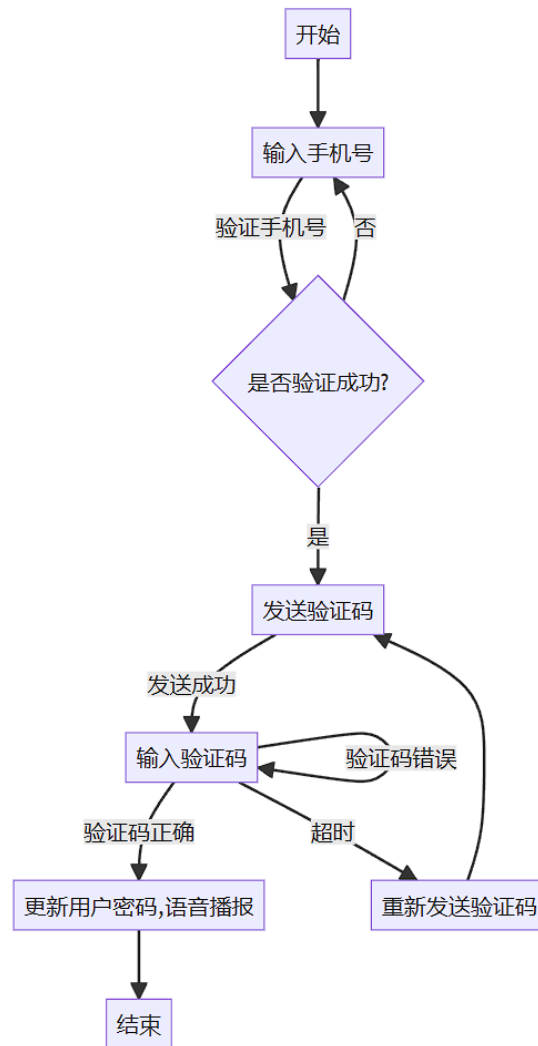


5. 忘记密码

该功能旨在解决用户忘记密码的问题。

用户输入手机号，先会利用正则表达式对手机号格式进行校验，并且在文件中查找是否存在此手机号，如何都通过了，则会模拟发送验证码，验证码设置了 60 秒的有效时间，通过当前时间减去发送时的时间计算出是否超过 60 秒，如果超过，则按回车键重新发送，否则显示用户名并输入新密码。

流程图：



6. 查询职工档案信息

(1) 自动全屏

当按“1”键进入查询职工档案信息功能时，会调用自定义函数 `fullScreen()` 时窗口最大化，便于显示数据。

(2) 分页查询

查询职工档案信息采用分页查询的方法。

通过自定义函数，将文件中的数据获取出来，放入一个 `EmployeeProfile` 类型的动态数组 `employeeProfiles` 中，计算分页参数页码 `pageNum` 和每页展示数 `pageSize`，获取对应的数据；

(3) 排序

工号、年龄和入职日期的升降序功能。

通过 `sort()` 函数和 `lambda` 函数对 `employeeProfiles` 进行排序, 实现正序和倒序功能。

(4) 页码和每页展示数

跳往页码、更改每页展示数。

通过修改 `pageNum` 和 `pageSize` 的值实现。

(5) 模糊查询

用户输入任何需要查询的内容, 系统会去文件中查询任何属性中包含了此查询内容的信息, 并且还计算了查询所需的时间。

通过自定义函数 `fuzzyQuery()` 实现此功能。其中在开始查询时记录开始时间, 查询结束之后记录结束时间, 再相减得到查询时间。在查询时, 会先通过 `regex_replace()` 函数将查询内容中的特殊字符进行转义, 防止在使用正则表达式时发生错乱。接下来通过 `regex_search()` 函数对查询内容和文件中的内容逐一进行匹配, 匹配成功则输出该条职工信息。

(6) 翻页功能

按 `A/←` 或 `D/→` 可以实现左右翻页功能。

通过 `_getch()` 获取用户按下的按键, 然后修改对应的 `pageNum` 实现该功能。

(7) 其余显示信息

在查询页面的尾部显示当前页、总页数、每页展示数和总职工数等信息。

1.按工号升序 2.按工号降序 3.按年龄升序 4.按年龄降序 5.按入职时间升序 6.按入职时间降序 7.前往页码 8.更改每页展示数 9.模糊查询 ESC键退出										
工号:	职工姓名:	身份证号:	性别:	年龄:	联系电话:	家庭地址:	学历:	职位:	入职日期:	所属部门:
2022040404	周正若	510104199608234524	女	27	15028182345	四川省成都市	硕士	人力资源专员	2023-10-12	人力资源部
2022050505	吴郭	31010119941010353X	男	29	18621101010	上海市黄浦区	本科	财务会计	2023-11-18	财务部
2022060606	张无忌	420102199312121817	男	30	13871212121	湖北省武汉市	硕士	研发工程师	2023-12-20	研发部
2022111111	刘洋	430103199709082723	女	26	18873109080	湖南省长沙市	本科	市场营销	2023-10-10	市场部
2022121212	孙丽	370203200306153247	女	20	15853261532	山东省青岛市	大专	客服代表	2023-11-20	客服部
2022123123	陈晨	330212200011110415	女	23	15957457111	浙江省宁波市	大专	UI设计师	2023-12-01	设计部
当前页: 1/11 6/页 向左: A/← 向右: D/→										
总职工数: 61										

7. 新增职工档案信息

在新增职工信息时, 会对职工的工号格式、身份证号格式、手机号格式以及其是否存在进行校验。信息输入完毕之后会将信息存储在文件中。

身份证号和手机号格式通过正则表达式进行校验; 校验是否存在时会通过自定义函数去文件中进行查询比对。

```

工号(长度为10位,前四位数为1956~当前年份之间):2022
工号格式有误!
工号(长度为10位,前四位数为1956~当前年份之间):1900212001
工号格式有误!
工号(长度为10位,前四位数为1956~当前年份之间):2025212001
工号格式有误!
工号(长度为10位,前四位数为1956~当前年份之间):2022212001
工号已存在!
工号(长度为10位,前四位数为1956~当前年份之间):2022212888
职工姓名:张三
身份证号(X为大写):362322222208091111
身份证号格式有误!
身份证号(X为大写):36232220040809111x
身份证号格式有误!
身份证号(X为大写):110101199507023512
身份证号已存在!
身份证号(X为大写):36232220040809111X
性别:男
年龄:23
联系电话:11122223333
手机号格式错误!
联系电话:18000001111
手机号已存在!
联系电话:18055556666
家庭地址:江西省南昌市
学历:本科
职位:软件开发
入职日期:2022-1-12
所属部门:开发岗|

```

8. 修改职工档案信息

先选择通过工号或者身份证号查询出职工信息

```

1.通过工号更新职工 2.通过身份证号更新职工 ESC键退出
|

```

信息查询出来之后会显示原来的值。工号不可修改,如果不进行修改按回车键即可,依旧会对职工的身份证号格式、手机号格式以及其是否存在进行校验,如图所示:

```

请输入工号： 2022212001
按回车键则保留旧数据

旧值： 2022212001
不可修改

旧值： 张伟
新值：

旧值： 362322200408121111
新值：

旧值： 男
新值：

旧值： 19
新值：

旧值： 18000001111
新值： 18055556666
手机号已存在！
请重新输入手机号： 18088889999

旧值： 江西省上饶市
新值： 江西省南昌市

旧值： 博士
新值：

旧值： 算法开发
新值：

旧值： 2024-4-2
新值：

旧值： 人工智能岗
新值：

工号为 2022212001 的员工记录已成功修改。
1.通过工号更新职工 2.通过身份证号更新职工 ESC键退出

```

通过自定义函数实现，首先创建一个临时文件，再去原文件中查询是否存在该工号(或身份证号)，如果存在该工号(或身份证号)，则令标识 **found** 为 **true**。如果某个职工信息不是要更新的职工，则写入临时文件，如果是要更新的职工则将其信息获取出来，将每个属性以逗号为标识符分开存入 **employeeVector** 中便于操作，再逐一显示原信息并输入新信息，输入完毕后将 **employeeVector** 中的信息拼接成一个新的字符串存入文件。最后对 **found** 进行判断，如果为 **true** 则将原文件删除，将临时文件命名为原文件的名称，否则删除临时文件。

在实现不进行修改按回车键即可时，是本项目的难点之一，需要对 C++ 的输入字符流有比较熟练的掌握，并且还要熟练使用 **Debug** 功能，否则很容易出现不理想的效果。

当不清空输入缓冲区时，前面的输入会影响到后面的效果，比如注释掉以下代码之后

```

for (int j = 0; j < employeeVector.size(); j++) {
    if (j == 0) {
        cout << employeeLanguage->retainOldVal << employeeVector[0] << endl
            << employeeLanguage->catNotUpdate << endl << endl;
        j++;
        // 清除输入缓冲区
        //cin.ignore();
    }
}

```

便会出现以下效果，将直接跳过姓名的修改。

```
请输入工号： 2022212001
按回车键则保留旧数据

旧值： 2022212001
不可修改

旧值： 张伟
新值：
旧值： 362322200408121111
新值： |
```

再比如，注释掉以下代码之后

```
for (int j = 0; j < employeeVector.size(); j++) {
    if (j == 0) {
        cout << employeeLanguage->ratainOldVal << employeeVector[0] << endl
            << employeeLanguage->catNotUpdate << endl << endl;
        j++;
        // 清除输入缓冲区
        cin.ignore();
    }
    cout << employeeLanguage->oldVal << employeeVector[j] << endl << employeeLanguage->newVal;
    // 读取用户输入的字符
    char input = cin.get();
    // 如果不是回车键
    if (input != '\n') {
        // 将读取的字符放回输入流
        //cin.unget();
        // 进行输入操作
        cin >> newE;
    }
}
```

可以看到，输入中文时无法识别到信息

```
189 string newE;
190 for (int j = 0; j < employeeVector.size(); j++) {
191     if (j == 0) {
192         cout << employeeLanguage->ratainOldVal << employeeVector[0] << endl
193             << employeeLanguage->catNotUpdate << endl << endl;
194         j++;
195         // 清除输入缓冲区
196         cin.ignore();
197     }
198     cout << employeeLanguage->oldVal << employeeVector[j] << endl << employeeLanguage->newVal;
199     // 读取用户输入的字符
200     char input = cin.get();
201     // 如果不是回车键
202     if (input != '\n') {
203         // 将读取的字符放回输入流
204         //cin.unget();
205         // 进行输入操作
206         cin >> newE;
207     }
208 }
209
210
211 请输入工号： 2022212001
212 按回车键则保留旧数据
213
214 旧值： 2022212001
215 不可修改
216
217 旧值： 张伟
218 新值： 张三丰
```

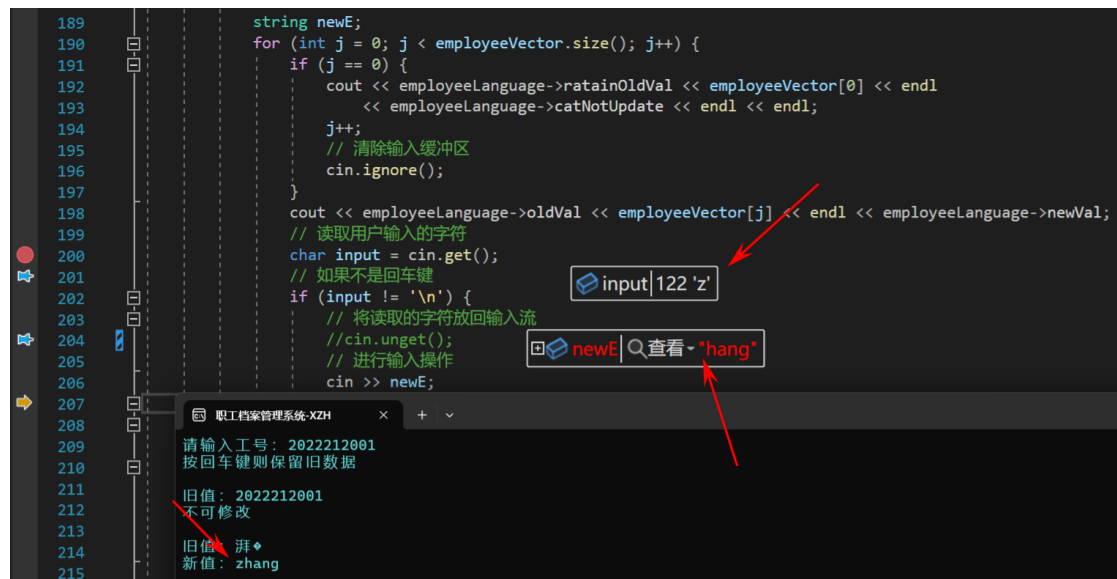
如果继续执行修改，最终保存的数据就会出现乱码

请输入工号：2022212001
按回车键则保留旧数据

旧值：2022212001
不可修改

旧值：湃◆
新值：

如果输入的是英文时将会丢失第一个字符

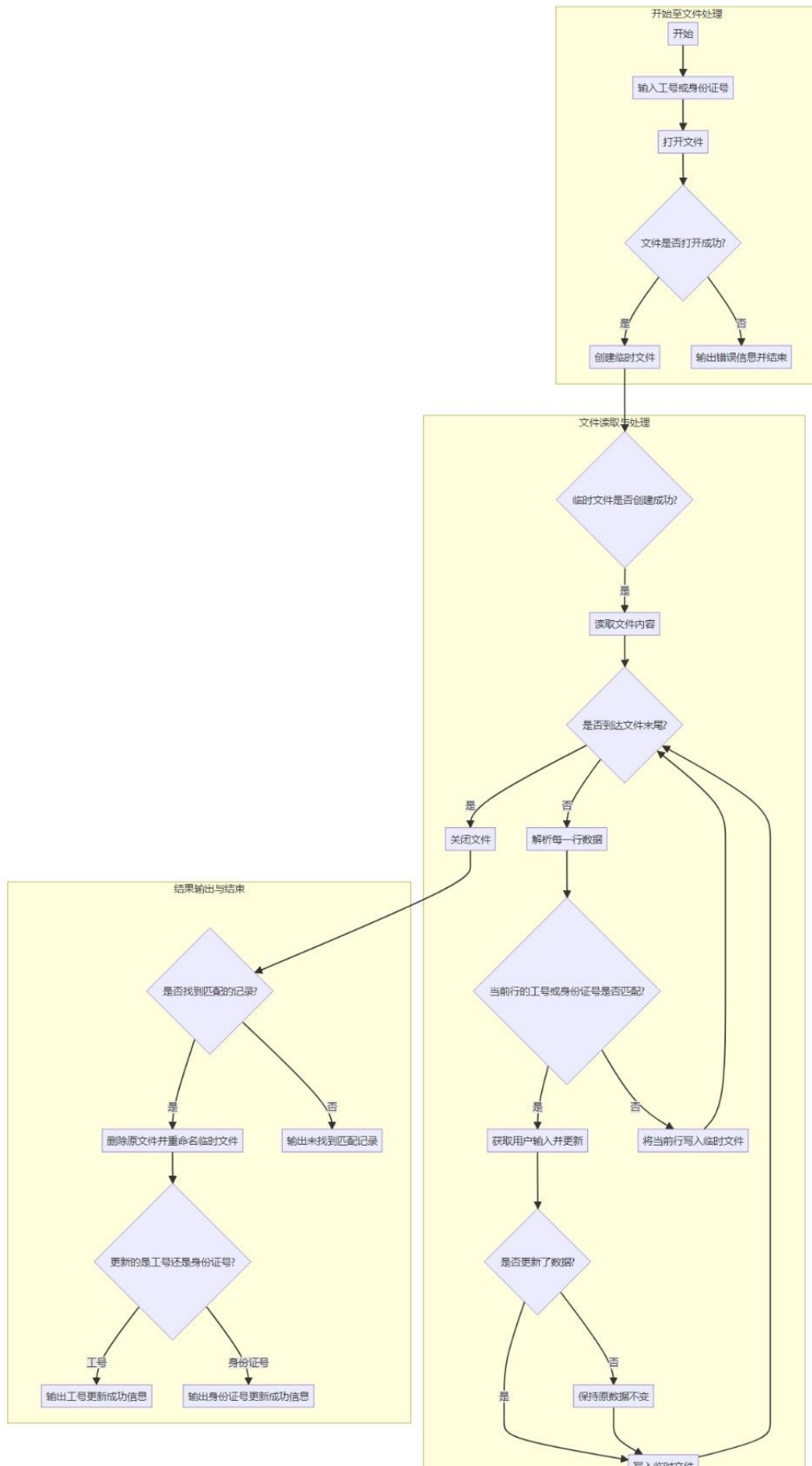


```
189     string newE;
190     for (int j = 0; j < employeeVector.size(); j++) {
191         if (j == 0) {
192             cout << employeeLanguage->retainOldVal << employeeVector[0] << endl
193                 << employeeLanguage->catNotUpdate << endl << endl;
194             j++;
195             // 清除输入缓冲区
196             cin.ignore();
197         }
198         cout << employeeLanguage->oldVal << employeeVector[j] << endl << employeeLanguage->newVal;
199         // 读取用户输入的字符
200         char input = cin.get();
201         // 如果不是回车键
202         if (input != '\n') {
203             // 将读取的字符放回输入流
204             // cin.unget();
205             // 进行输入操作
206             cin >> newE;
207         }
208     }
209     // 职工档案管理系统-XZH
210     请输入工号：2022212001
211     按回车键则保留旧数据
212     旧值：2022212001
213     不可修改
214     旧值：湃◆
215     新值：zhang
```

input|122 'z'

newE 查看 "hang"

流程图：



9. 删除职工档案信息

通过工号和身份证号进行删除。

通过自定义函数实现，首先创建一个临时文件，先去原文件中查询是否存在该工号(或身份证号)，如果存在该工号(或身份证号)，则令标识 **found** 为 **true**。如果某个职工信息不是要删除的职工，则写入临时文件，如果是要删除的职工则不写入临时文件。最后对 **found** 进行判断，如果为 **true** 则将原文件删除，将临时文件命名为原文件的名称，否则删除临时文件。

```
请输入工号：2022212881
未找到工号为 2022212881 的员工记录。
1.通过工号删除 2.通过身份证号删除 ESC键退出
```

```
请输入工号：2022212888
工号为 2022212888 的员工记录已成功删除。
1.通过工号删除 2.通过身份证号删除 ESC键退出
```

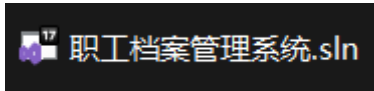
考虑到删除全部数据实现起来相对简单（只需创建临时文件，删除原文件，再将临时文件改名为原文件名即可），但是相对来说不太安全，容易将数据全部删除，故本项目不实现删除全部员工操作功能。

五、运行环境及使用说明

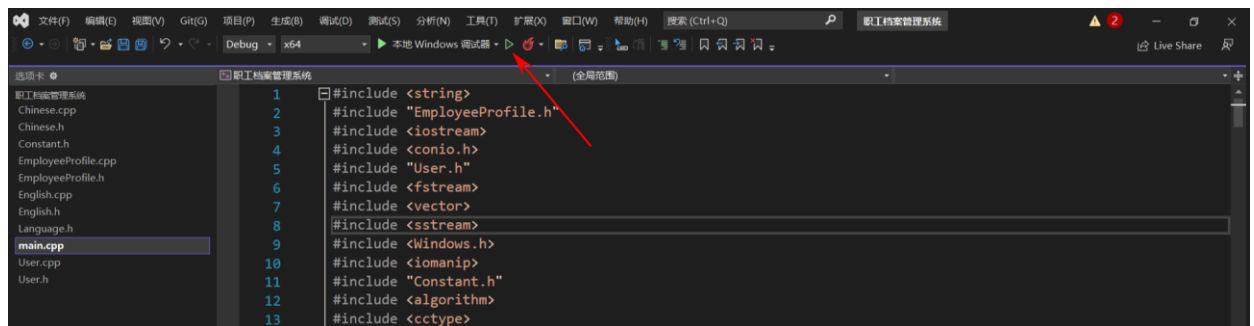
操作系统：Windows 11

运行软件：Microsoft Visual Studio 2022

本项目主函数为 **main()** 函数。运行本项目，用 Microsoft Visual Studio 2022 打开“职工档案管理系统.sln”文件



直接运行即可



在运行本项目时，最好确保当前目录下存在“user.txt”和“employee.txt”文件，并且在当前目录下要存在 video 文件夹，将音频文件放入其中。

六、项目总结与展望

在“职工档案管理系统”的开发过程中，我深刻领悟到 C++ 面向对象编程的精髓与实践应用的价值。本课程设计项目是通信工程专业学生综合能力培养的重要环节，它要求我在掌握面向对象程序设计的基础之上，通过具体应用案例，将理论知识转化为实用技能。

项目伊始，我明确了职工档案管理系统目标——实现职工档案信息的高效管理，包括显示、录入、删除、修改、排序及查询等功能。为了增强系统实用性与用户体验，我引入了中英双语支持、背景音乐与语音播报等特色功能，不仅丰富了系统的交互性，也提升了用户使用的便捷性和趣味性。

在功能实现上，我注重细节与用户体验。注册流程中，通过正则表达式验证手机号格式，确保信息准确无误；登录功能的校验机制，有效保障了账户安全；忘记密码功能通过验证码验证，既保证安全性又兼顾用户便利。查询职工档案信息时，自动全屏与分页查询的结合，让大量数据的浏览变得井然有序；模糊查询与排序功能，极大地提高了数据检索的效率和准确性。

新增、修改与删除职工档案信息时，我采用了严谨的数据校验与处理策略，保证信息的完整性与一致性。特别是在修改信息时，对输入流的精准控制，避免了因缓冲区未清空导致的数据混乱。

在运行环境配置上，我选择了 Windows 11 操作系统与 Microsoft Visual Studio 2022 作为开发工具，确保了项目在主流环境下的稳定运行。项目中涉及的“user.txt”与“employee.txt”文件以及音频文件的妥善处理，为系统的正常运行提供了必要的数据库支持。

项目总结阶段，我回顾整个开发历程，一次又一次的困难，一次又一次的查

阅资料、解决 Bug，深感每一次挑战的克服都是个人技能提升的契机。本项目不仅是对我学习《C++面向对象程序设计》课程结果的检验，也是对我解决复杂问题和创新思维的全方位锻炼。

我期望，“职工档案管理系统”在未来可以结合 EasyX 图形库或 Qt 框架等，实现更加直观、友好的用户界面。更进一步可以结合 HTML、CSS 和 JavaScript 等相关的前端开发技术，可以将“职工档案管理系统”从桌面应用拓展至网络平台，让本系统变得更加实用和灵活。

最后，我期待通过此次项目，向涂老师展现我的努力与成果。

参考文献

- [1] 邵兰洁, 马睿, 徐海云, 母俐丽. (2020). C++面向对象程序设计（第二版）. 清华大学出版社.
- [2] 陈泓文, 邹昶, 聂国健, 等. 基于面向对象思想的软件系统分析与设计[J]. 电子产品可靠性与环境试验, 2020, 38(06):24-28.
- [3] 余桂强. 成都市六医院职工档案管理系统设计与实现[D]. 电子科技大学, 2018.
- [4] 田婷. 贵医附院职工档案管理系统的研究与分析[D]. 云南大学, 2015.
- [5] 董引娣. 职工档案管理系统设计[J]. 软件导刊, 2013, 12(12):87-88.

附录

1. Main.cpp

```
#include <string>
#include "EmployeeProfile.h"
#include <iostream>
#include <conio.h>
#include "User.h"
#include <fstream>
#include <vector>
#include <sstream>
#include <Windows.h>
#include <iomanip>
#include "Constant.h"
```

```
#include <algorithm>
#include <cctype>
#include <regex>
#include <functional>
#include <chrono>
#include "Language.h"
#include "Chinese.h"
#include "English.h"
#include <memory>
#pragma comment(lib, "Winmm.lib")
using namespace std;
string EnString = "_ENGLISH";
string resultString = "";
// 工厂函数，根据需要返回相应的Language实例
shared_ptr<Language> createLanguage(const string& lang) {
    if (lang == "_ENGLISH") {
        return make_shared<English>();
    } else {
        // 默认中文或其他逻辑
        return make_shared<Chinese>();
    }
}
// shared_ptr允许多个对象共享同一个资源的所有权，
// 当最后一个指向该资源的shared_ptr销毁时，资源会被自动释放
shared_ptr<Language> language;
/**
 * @author XZH
 */
bool login();
void myRegister();
void forgetPassword();
void selectMusic();
void menu();
void displayAllProfile();
void insertEmployeeProfile();
void updateEmployeeProfile(string filename);
void updateEmployeeProfileById(string filename);
void updateEmployeeProfileByIdNumber(string filename);
void deleteEmployeeProfile();
void deleteEmployeeProfileById(string filename);
void deleteEmployeeProfileByIdNumber(string filename);
void fuzzyQuery(vector<EmployeeProfile> employeeProfiles);
```

```
vector<EmployeeProfile> loadEmployeeProfiles(const string& filename);
EmployeeProfile createProfileFromLine(const std::string& line);
void fullScreen();
void closeBlackWindow();
void videoPath(wstring basePath);
void tableTitle();
// 是否选择过语言
bool selectLanguage = false;
int main() {
    while (1) {
        // 如果还没选择语言
        if (!selectLanguage) {
            while (1) {
                cout << "请选择语言\tPlease select language" << endl
                     << "1. 中文\t\t1.Chinese" << endl
                     << "2. 英文\t\t2.English" << endl;
                int languageChoice = _getch();
                if (languageChoice == '1') {
                    std::system("cls");
                    selectLanguage = true;
                    // 选择语言对象
                    language = createLanguage(resultString);
                    // 是否开启背景音乐
                    selectMusic();
                    std::system("cls");
                    videoPath(L"video\\welcome");
                    break;
                }
                else if (languageChoice == '2') {
                    std::system("cls");
                    resultString = EnString;
                    selectLanguage = true;
                    // 选择语言对象
                    language = createLanguage(resultString);
                    // 是否开启背景音乐
                    selectMusic();
                    std::system("cls");
                    videoPath(L"video\\welcome");
                    break;
                }
            }
        }
        else {
            std::system("cls");
        }
    }
}
```

```
        cout << KEY_ERROR << endl;
        cout << KEY_ERROR_ENG << endl << endl;
    }
}

// 将语言对象传递给User
User user(language);
// 将语言对象传递给Employee
EmployeeProfile employee(language);
// 设置黑窗口标题
SetConsoleTitleA("职工档案管理系统-XZH");
// 设置黑窗口和字体颜色
system("color f0");
// 显示日期
system("date /T");
// 显示时间
system("TIME /T");
cout << language->loginAndRegisterPage() << endl;
char first;
first = _getch();
switch (first) {
case '1':
    std::system("cls");
    cout << language->welcomeLogin << endl;
    // 如果登录成功, 进入主界面
    if (login()) {
        // 设置黑窗口和字体颜色
        system("color 0B");
        menu();
        return 0;
    }
    break;
case '2':
    std::system("cls");
    cout << language->welcomeRegister << endl;
    myRegister();
    break;
case '3':
    std::system("cls");
    cout << language->forgetPassword << endl;
    forgetPassword();
    break;
```

```
case 27:
    cout << language->isExit << endl << language->yesOrNo;
    if (_getch() == '1') {
        cout << language->exitSuccess << endl;
        videoPath(L"video\\bye");
        closeBlackWindow();
        // 用return 0 的话在meun()中调用main()之后还会再执行meun()
        // 虽然执行了关闭窗口函数，但是如果用户此时鼠标正在移动窗口
        // 则不会直接关闭窗口
        exit(0);
    }
    std::system("cls");
    break;
default:
    system("cls");
    cout << language->KEY_ERROR << endl;
}
}
return 0;
}
// 登录
bool login() {
    string name, password;
    string filename = USER_FILENAME;
    while (1) {
        cout << language->inputUserName;
        cin >> name;
        // 检查用户名是否存在
        bool nameExists = User::isUsernameExists(name, filename);
        if (!nameExists) {
            //std::system("cls");
            cout << language->userNameNotExist << endl;
            return false;
        }
        cout << language->inputPassword;
        password = User::inputPassword();
        User user(name, password);
        if (user.isPasswordValid(filename)) {
            std::system("cls");
            cout << language->loginSuccess << endl;
            videoPath(L"video\\login");
            return true;
        }
    }
}
```

```
    }
    else {
        cout << language->userNameOrPasswordError << endl;
    }
}

}

// 注册
void myRegister() {
    string name, password, confirmPassword, phone;
    string filename = USER_FILENAME;
    while (1) {
        cout << language->inputUserName;
        cin >> name;
        // 检查用户名是否存在
        bool nameExists = User::isUsernameExists(name, filename);
        if (nameExists) {
            cout << language->userNameAlreadyExist << endl;
        }
        else {
            break;
        }
    }
    while (1) {
        cout << language->inputPhone;
        cin >> phone;
        // 如果手机号格式不正确
        if (!EmployeeProfile::verifyPhone(phone)) {
            continue;
        }
        // 检查手机号是否存在
        bool phoneExists = User::isPhoneExists(phone, filename);
        if (phoneExists) {
            cout << language->phoneAlreadyExist << endl;
        }
        else {
            break;
        }
    }
    while (1) {
        cout << language->inputPassword;
        password = User::inputPassword();
```



```
        cout << language->inputConfirmPassword;
        confirmPassword = User::inputPassword();
        if (password != confirmPassword) {
            cout << language->passwordDifferent << endl;
        }
        else {
            break;
        }
    }
    User user(name, password, phone);
    std::system("cls");
    // 保存用户信息
    user.saveUserToFile(filename);
    cout << language->registerSuccess << endl;
    videoPath(L"video\\register");
}
// 选择音乐
void selectMusic() {
    const wstring fileExtension = L".mp3 repeat";
    const wstring basePath = L"play video\\";
    cout << language->music << endl << language->yesOrNo2 << endl;
    int key = _getch();
    // 不开启背景音乐
    if (key == '2') {
        return;
    }
    int i;
    // 设置随机数生成器的种子
    srand(time(0));
    while (1) {
        i = rand() % 10;
        if (i <= 5 && i > 0) {
            break;
        }
    }
    // 拼接字符串
    wstring filePath = basePath + to_wstring(i) + fileExtension;
    mciSendString(filePath.c_str(), NULL, 0, NULL);
}
// 忘记密码
void forgetPassword() {
    string phone, code, VerificationCode;
```

```

    chrono::high_resolution_clock::time_point startTime, endTime;
    while (1) {
        cout << language->inputPhone;
        cin >> phone;
        if (!EmployeeProfile::verifyPhone(phone)) {
            continue;
        }
        sendVerificationCode:
        VerificationCode = User::findPhoneAndSendVerificationCode(phone,
USER_FILENAME);
        // 记录开始时间
        startTime = chrono::high_resolution_clock::now();
        if (VerificationCode != "") {
            break;
        }
    }
    cout << language->inputCode;
    while (1) {
        cin >> code;
        // 记录结束时间
        endTime = chrono::high_resolution_clock::now();
        if (chrono::duration_cast<chrono::seconds>(endTime - startTime).count() >
3) {
            cout << language->overtime << endl << language->pressEnter << endl;
            int i = _getch();
            if (i == 13) {
                goto sendVerificationCode;
            }
            break;
        }
        if (VerificationCode == code) {
            User::updateUserByPhone(USER_FILENAME, phone);
            videoPath(L"video\\password");
            break;
        }
        cout << language->codeError;
    }
}
// 功能菜单
void menu() {
    while (1) {
        language->menu();
    }
}

```

```
    string filename = EMPLOYEE_FILENAME;
    char i;
    i = _getch();
    switch (i)
    {
    case 27:
        cout << language->isExit << endl << language->yesOrNo;
        if (_getch() == '1') {
            std::system("cls");
            main();
        }
        std::system("cls");
        break;
    case '1':
        std::system("cls");
        displayAllProfile();
        break;
    case '2':
        std::system("cls");
        insertEmployeeProfile();
        break;
    case '3':
        std::system("cls");
        updateEmployeeProfile(filename);
        break;
    case '4':
        std::system("cls");
        deleteEmployeeProfile();
        break;
    default:
        std::system("cls");
        cout << language->KEY_ERROR << endl;
        break;
    }
}

// 显示所有职工信息，并提供排序、模糊查询等功能
void displayAllProfile() {
    // 窗口最大化
    fullScreen();
    vector<EmployeeProfile> employeeProfiles =
```

```

loadEmployeeProfiles(EMPLOYEE_FILENAME);

bool flag = true;
while (flag) {
    cout << language->selectFunction << endl << endl;
    // 输出表头
    tableTitle();
    int startRow = pageNum > 0 ? (pageNum - 1) * pageSize : 0;
    int endRow = startRow + pageSize * (pageNum > 0 ? 1 : 0);
    int total = employeeProfiles.size();
    // 计算出来之后向上取整
    int sumNum = static_cast<int>(ceil(static_cast<double>(total) / pageSize));
    // 输出每一个数据
    for (int i = startRow; i < endRow && i < total; i++) {
        cout << employeeProfiles[i] << endl;
    }
    cout << language->currentPage << pageNum << "/" << sumNum << "\t" <<
    pageSize << language->page <<
        language->leftOrRight << endl
        << language->totalEmployee << total;
    int i = _getch();
    switch (i) {
    case '1':
        std::system("cls");
        sort(employeeProfiles.begin(), employeeProfiles.end(),
        [](EmployeeProfile& a, EmployeeProfile& b) {
            // 按工号升序
            return a.getId() < b.getId();
        });
        break;
    case '2':
        std::system("cls");
        sort(employeeProfiles.begin(), employeeProfiles.end(),
        [](EmployeeProfile& a, EmployeeProfile& b) {
            // 按工号降序
            return a.getId() > b.getId();
        });
        break;
    case '3':
        std::system("cls");
        sort(employeeProfiles.begin(), employeeProfiles.end(),
        [](EmployeeProfile& a, EmployeeProfile& b) {
            // 按年龄升序

```

```
        return a.getAge() < b.getAge();
    });
    break;
case '4':
    std::system("cls");
    sort(employeeProfiles.begin(), employeeProfiles.end(),
    [](EmployeeProfile& a, EmployeeProfile& b) {
        // 按年龄序
        return a.getAge() > b.getAge();
    });
    break;
case '5':
    std::system("cls");
    sort(employeeProfiles.begin(), employeeProfiles.end(),
    [](EmployeeProfile& a, EmployeeProfile& b) {
        // 按入职时间升序
        return a.getHireDate() < b.getHireDate();
    });
    break;
case '6':
    std::system("cls");
    sort(employeeProfiles.begin(), employeeProfiles.end(),
    [](EmployeeProfile& a, EmployeeProfile& b) {
        // 按入职时间降序
        return a.getHireDate() > b.getHireDate();
    });
    break;
case '7': {
    cout << language->inputPageNum;
    string inputStr;
    cin >> inputStr;
    std::system("cls");
    try {
        // 检测输入是否可以转成在范围内的数字
        int input = stoi(inputStr);
        // 如果输入页码小于等于0则为第一页
        if (input <= 0) {
            pageNum = 1;
        }
        // 如果大于最大页码最为最后一页
        else if (input > sumNum) {
            pageNum = sumNum;
        }
    }
}
```

```
    }
    else {
        pageNum = input;
    }
}
catch (const invalid_argument& e) {
    // 捕获输入不是数字的异常
    cerr << language->inputNum << endl;
}
catch (const out_of_range& e) {
    // 转换后的数字超出int范围
    cerr << language->numTooLong << endl;
}
break;
}
case '8':
    cout << language->inputNumPerPage;
    cin >> pageSize;
    std::system("cls");
    // 回到第一页
    pageNum = 1;
    break;
case '9':
    fuzzyQuery(employeeProfiles);
    std::system("pause");
    std::system("cls");
    break;
case 27: // ESC键
    std::system("cls");
    flag = false;
    break;
case 77: // 右键
    std::system("cls");
    if (pageNum < sumNum) {
        pageNum++;
    }
    break;
case 100: // D键
    std::system("cls");
    if (pageNum < sumNum) {
        pageNum++;
    }
}
```

```
        break;
    case 75: // 左键
        std::system("cls");
        if (pageNum > 1) {
            pageNum--;
        }
        break;
    case 97: // A键
        std::system("cls");
        if (pageNum > 1) {
            pageNum--;
        }
        break;
    default:
        std::system("cls");
        //cout << i;
        cout << language->KEY_ERROR << endl;
    }
}

// 重置为第一页
pageNum = PAGE_NUM;
// 重置每页展示数
pageSize = PAGE_SIZE;
}

// 新增职工
void insertEmployeeProfile() {
    EmployeeProfile e;
    cin >> e;
    e.saveEmployeeToFile(EMPLOYEE_FILENAME);
}

// 修改职工信息
void updateEmployeeProfile(string filename) {
    while (1) {
        cout << language->updateByIdOrIdNumber << endl;
        char i = _getch();
        if (i == '1') {
            std::system("cls");
            updateEmployeeProfileById(EMPLOYEE_FILENAME);
        }
        else if (i == '2') {
            std::system("cls");
            updateEmployeeProfileByIdNumber(EMPLOYEE_FILENAME);
        }
    }
}
```

```
    }
    else if (i == 27) {
        std::system("cls");
        break;
    }
    else {
        cout << language->inputOneOrTwo << endl;
    }
}
}

// 通过工号更新职工信息
void updateEmployeeProfileById(string filename) {
    EmployeeProfile::updateProfileById(filename);
}

// 通过身份证号更新职工信息
void updateEmployeeProfileByIdNumber(string filename) {
    EmployeeProfile::updateProfileByIdNumber(filename);
}

// 删除员工
void deleteEmployeeProfile() {
    while (1) {
        cout << language->deleteByIdOrIdNumber << endl;
        char i = _getch();
        if (i == '1') {
            std::system("cls");
            deleteEmployeeProfileById(EMPLOYEE_FILENAME);
        }
        else if (i == '2') {
            std::system("cls");
            deleteEmployeeProfileByIdNumber(EMPLOYEE_FILENAME);
        }
        else if (i == 27) {
            std::system("cls");
            break;
        }
        else {
            cout << language->inputOneOrTwo << endl;
        }
    }
}

// 通过工号删除
void deleteEmployeeProfileById(string filename) {
```



```

    EmployeeProfile::deleteProfileById(filename);
}
// 通过身份证号删除
void deleteEmployeeProfileByIdNumber(string filename) {
    EmployeeProfile::deleteProfileByIdNumber(filename);
}
// 模糊查询
void fuzzyQuery(vector<EmployeeProfile> employeeProfiles) {
    chrono::high_resolution_clock::time_point startTime, endTime;
    string content;
    // 记录查询结果数
    int count = 0;
    cout << language->inputSelectContent;
    cin >> content;
    std::system("cls");
    // 转义content中的特殊正则表达式字符
    string escapedContent = regex_replace(content, regex(R"([\^$.|?*(\)])"),
R"(\&$)");
    regex reg(escapedContent, regex_constants::icase);
    // 输出表头
    tableTitle();
    // 记录开始时间
    startTime = chrono::high_resolution_clock::now();
    // 逐一匹配
    for (int i = 0; i < employeeProfiles.size(); i++) {
        if (regex_search(employeeProfiles[i].getId(), reg) ||
            regex_search(employeeProfiles[i].getName(), reg) ||
            regex_search(employeeProfiles[i].getIdNumber(), reg) ||
            regex_search(employeeProfiles[i].getGender(), reg) ||
            regex_search(to_string(employeeProfiles[i].getAge()), reg) ||
            regex_search(employeeProfiles[i].getPhoneNumber(), reg) ||
            regex_search(employeeProfiles[i].getAddress(), reg) ||
            regex_search(employeeProfiles[i].getEducation(), reg) ||
            regex_search(employeeProfiles[i].getPosition(), reg) ||
            regex_search(employeeProfiles[i].getHireDate(), reg) ||
            regex_search(employeeProfiles[i].getDepartment(), reg)) {
            // 打印匹配到的员工信息
            cout << employeeProfiles[i] << endl;
            count++;
        }
    }
}
// 记录结束时间

```

```

    endTime = chrono::high_resolution_clock::now();
    cout << language->selectTime
        << chrono::duration_cast<chrono::milliseconds>(endTime - startTime).count()
    << " 毫秒" << endl
        << language->totalEmployee << count << endl;
}

// 从文件中获取每一行的内容，并构建成一个EmployeeProfile对象数组
vector<EmployeeProfile> loadEmployeeProfiles(const string& filename) {
    vector<EmployeeProfile> profiles;
    ifstream inFile(filename);
    if (inFile.is_open()) {
        std::string line;
        while (std::getline(inFile, line)) {
            // 确保行不为空
            if (!line.empty()) {
                EmployeeProfile profile = createProfileFromLine(line);
                profiles.push_back(profile);
            }
        }
        inFile.close();
    }
    else {
        cerr << language->canNotOpen << filename << std::endl;
    }
    return profiles;
}

// 从一行数据中获取每个属性值，并构建EmployeeProfile对象
EmployeeProfile createProfileFromLine(const std::string& line) {
    istringstream iss(line);
    // 定义 EmployeeProfile 包含的成员变量
    string id, name, idNumber, gender, ageStr, phoneNumber, address, education,
position, hireDate, department;
    int age;
    // 每行中的每个数据以逗号分隔，解析字符串
    getline(iss, id, ',');
    getline(iss, name, ',');
    getline(iss, idNumber, ',');
    getline(iss, gender, ',');
    // 对于整数非字符串类型，需要额外的转换
    getline(iss, ageStr, ',');
    getline(iss, phoneNumber, ',');
    getline(iss, address, ',');

```

```
getline(iss, education, ',');
getline(iss, position, ',');
getline(iss, hireDate, ',');
getline(iss, department);
// 将字符串ageStr转化为int类型age
age = stoi(ageStr);
// 创建 EmployeeProfile 对象
return EmployeeProfile(id, name, idNumber, gender, age, phoneNumber, address,
education, position, hireDate, department);
}
// 窗口最大化
void fullScreen() {
    // 按下Alt键
    keybd_event(VK_MENU, 0, 0, 0);
    // 按下空格键
    keybd_event(VK_SPACE, 0, 0, 0);
    // 按下X键
    keybd_event(0x58, 0, 0, 0);
    // 释放X键
    keybd_event(0x58, 0, KEYEVENTF_KEYUP, 0);
    // 释放空格键
    keybd_event(VK_SPACE, 0, KEYEVENTF_KEYUP, 0);
    // 释放Alt键
    keybd_event(VK_MENU, 0, KEYEVENTF_KEYUP, 0);
}
// 关闭窗口
void closeBlackWindow() {
    cout << language->close;
    Sleep(1000);
    cout << ".";
    Sleep(1000);
    cout << ".";
    Sleep(1000);

    // 按下Alt键
    keybd_event(VK_MENU, 0, 0, 0);
    // 按下F4键
    keybd_event(VK_F4, 0, 0, 0);
    // 释放Alt键
    keybd_event(VK_MENU, 0, KEYEVENTF_KEYUP, 0);
    // 释放F4键
    keybd_event(VK_F4, 0, KEYEVENTF_KEYUP, 0);
}
```

```
}
// 播放不同语言的音频
void videoPath(wstring basePath) {
    const wstring fileExtension = L".wav";
    // 拼接字符串
    wstring filePath = basePath
        + wstring(resultString.begin(), resultString.end())
        + fileExtension;
    // 播放音频
    PlaySound(filePath.c_str(), NULL, SND_FILENAME | SND_ASYNC);
}
// 打印表头
void tableTitle() {
    const int SHORT_WIDTH = 10; // 短列宽
    const int MIDDLE_WIDTH = 15; // 中列宽
    const int LONG_WIDTH = 20; // 长列宽
    const int LINE_LENGTH = 155; // 分割线长度
    // 输出表头
    language->tableTitle();
    cout << string(LINE_LENGTH, '-') << endl;
}
```

2. User.h

```
#pragma once
#include <string>
#include "Language.h"
#include <memory>

using namespace std;
/**
 * @author XZH
 */
class User
{
private:
    // 用户名
    string name;
    // 密码
    string password;
    // 手机号
```

```

    string phone;
    // userLanguage是一个智能指针，用于管理Language类的对象
    static shared_ptr<Language> userLanguage;
    // 通过第i项内容查询并更新
    static void updateUserByI(const string& filename, const string& phone, int i);
public:
    // 无参构造函数
    User() = default;
    // 有参构造函数
    User(string na, string pa, string ph);
    User(string na, string pa);
    // 用于初始化语言的构造函数
    User(shared_ptr<Language>& langPtr);

    // 成员变量对应的get函数
    string getName();
    string getPassword();
    // 将用户信息保存至文件中
    void saveUserToFile(const string& filename);
    // 判断用户名和密码是否匹配
    bool isPasswordValid(const string& filename);
    // -----static-----
    // 判断是否存在该用户名
    static bool isUsernameExists(const std::string& username, const std::string&
filename);
    // 判断是否存在该手机号
    static bool isPhoneExists(const std::string& phone, const string& filename);
    // 手机号存在时发送验证码
    static string findPhoneAndSendVerificationCode(const string& phone, const
string& filename);
    // 根据手机号更新职工
    static void updateUserByPhone(const string& filename, const string& phone);
    // 输入密码
    static string inputPassword();
};

```

3. User.cpp

```

#include "User.h"
#include <fstream>
#include <iostream>

```

```
#include <string>
#include "Constant.h"
#include <sstream>
#include <vector>
#include "MD5.h"
#include <conio.h>
#include <windows.h>

using namespace std;
/**
 * @author XZH
 */
// 初始化静态成员变量
shared_ptr<Language> User::userLanguage = nullptr;
// 通过第i项内容查询并更新
void User::updateUserByI(const string& filename, const string& phone, int i)
{
    ifstream inFile(filename);
    if (!inFile.is_open()) {
        cerr << userLanguage->canNotOpen << endl;
        return;
    }
    // 创建临时文件来保存修改后的数据
    string tempFilename = "temp_" + filename;
    ofstream outFile(tempFilename);
    if (!outFile.is_open()) {
        cerr << userLanguage->canCreateTemp << endl;
        inFile.close();
        return;
    }
    string line;
    while (getline(inFile, line)) {
        istringstream iss(line);
        string currentProperty;
        // 读取每行的第i个字段（工号）
        int tempI = i;
        while (tempI--) {
            getline(iss, currentProperty, ',');
        }
        if (currentProperty != phone) {
            // 如果当前行的属性不是要修改的属性，则写入临时文件
            outFile << line << endl;
        }
    }
}
```

```
}
else {
    // 此处进行break的话会导致要删除职工后面的用户没有写入新文件
    // 使用vector来动态存储分割后的字符串
    vector<string> userVector;
    stringstream ss(line);
    string item;
    // 使用getline按逗号分割字符串，直到读取完所有元素
    while (getline(ss, item, ',')) {
        // 将分割得到的元素添加到vector中
        userVector.push_back(item);
    }
    cout << userLanguage->userName << userVector[USER_NAME_SERIAL_NUMBER - 1]
<< endl;

    string password, confirmPassword;
    while (1) {
        cout << userLanguage->inputNewPassword;
        password = inputPassword();
        cout << userLanguage->inputConfirmPassword;
        confirmPassword = inputPassword();
        if (password != confirmPassword) {
            cout << userLanguage->passwordDifferent << endl;
        }
        else {
            break;
        }
    }

    userVector[USER_PASSWORD_SERIAL_NUMBER - 1] = password;
    // 拼接新的用户信息
    string newLine = "";
    for (int j = 0; j < userVector.size(); j++) {
        newLine += userVector[j] + ",";
    }
    // 写入文件
    outFile << newLine << endl;
}
}
inFile.close();
outFile.close();
```

```
// 删除原文件
remove(filename.c_str());
// 将临时文件重命名为原文件名
rename(tempFilename.c_str(), filename.c_str());
cout << userLanguage->phone << phone << userLanguage->updatePasswordSuccess <<
endl;
}
User::User(string na, string pa, string ph) : name(na), password(pa), phone(ph)
{
}
User::User(string na, string pa) : name(na), password(pa)
{
}
User::User(shared_ptr<Language>& langPtr)
{
    userLanguage = langPtr;
}

string User::getName() {
    return name;
}
string User::getPassword() {
    return password;
}

// 将用户信息保存至文件中
void User::saveUserToFile(const string& filename) {
    ofstream outFile(filename, ios_base::app);
    if (!outFile.is_open()) {
        cerr << userLanguage->canNotOpen << filename << endl;
        return;
    }
    // 将用户名和密码写入文件，中间用逗号分开，便于之后解析
    outFile << name << "," << password << "," << phone << endl;
    outFile.close();
    cout << userLanguage->saveUser << filename << endl;
}
// 判断用户名和密码是否匹配
bool User::isPasswordValid(const string& filename)
{
    ifstream inFile(filename);
    if (!inFile.is_open()) {
```



```

        cerr << userLanguage->canNotOpen << filename << endl;
        // 文件不存在或无法打开, 无法验证
        return false;
    }
    string line;
    while (getline(inFile, line)) {
        // 查找第一个逗号的位置
        size_t pos1 = line.find(',');
        if (pos1 != string::npos) {
            // 获取第一个数据 (用户名)
            string storedUsername = line.substr(0, pos1);
            // 从第一个逗号之后开始查找第二个逗号的位置
            size_t pos2 = line.find(',', pos1 + 1);
            if (pos2 != string::npos) {
                // 获取第二个数据 (密码)
                string storedPassword = line.substr(pos1 + 1, pos2 - pos1 - 1);
                // 校验用户名和密码
                if (storedUsername == name && storedPassword == password) {
                    inFile.close();
                    // 用户名和密码匹配
                    return true;
                }
            }
        }
    }
    inFile.close();
    // 未找到匹配的用户名密码组合
    return false;
}

// 判断是否存在该用户名
bool User::isUsernameExists(const string& username, const string& filename) {
    ifstream inFile(filename);
    if (!inFile.is_open()) {
        cerr << userLanguage->canNotOpen << filename << endl;
        // 文件不存在或无法打开, 视为用户名不存在
        return false;
    }
    string line;
    while (getline(inFile, line)) {
        size_t pos = line.find(',');
        if (pos != string::npos && line.substr(0, pos) == username) {
            inFile.close();

```

```
        // 用户名已存在
        return true;
    }
}
inFile.close();
// 读取完所有行，未找到匹配的用户名
return false;
}
// 判断是否存在该手机号
bool User::isPhoneExists(const string& phone, const string& filename)
{
    ifstream inFile(filename);
    if (!inFile.is_open()) {
        cerr << userLanguage->canNotOpen << filename << endl;
        return false; // 文件不存在或无法打开，视为用户名不存在
    }
    string line;
    while (getline(inFile, line)) {
        istringstream iss(line);
        string currentProperty;
        // 读取每行的phone属性
        int tempI = USER_PHONE_SERIAL_NUMBER;
        while (tempI--) {
            getline(iss, currentProperty, ',');
        }
        if (currentProperty == phone) {
            inFile.close();
            // 手机号已存在
            return true;
        }
    }
    inFile.close();
    // 读取完所有行，未找到匹配的用户名
    return false;
}
// 手机号存在时发送验证码
string User::findPhoneAndSendVerificationCode(const string& phone, const string&
filename)
{
    if (isPhoneExists(phone, filename)) {
        srand(time(0));
        // 生成一个四位数的验证码
```

```
int verificationCode = rand() % 9000 + 1000;
cout << userLanguage->wait << endl;
Sleep(1000);
cout << userLanguage->sending << endl;
Sleep(1000);
cout << userLanguage->sended << phone << userLanguage->sendCode <<
    verificationCode << userLanguage->timeValid << endl;
return to_string(verificationCode);
}
else {
    cout << userLanguage->phoneNotRegister << endl;
}
// 未找到对应用户
return "";
}
// 根据手机号更新职工
void User::updateUserByPhone(const string& filename, const string& phone)
{
    updateUserByI(filename, phone, USER_PHONE_SERIAL_NUMBER);
}
// 输入密码
string User::inputPassword() {
    char password[21]; // 20位密码加上字符串结束符
    int index = 0;
    while (1) {
        char ch;
        ch = _getch();
        if (ch == 8) { // 退格键
            if (index != 0) {
                cout << "\b \b"; // 删除一个字符
                index--;
            }
        }
        else if (ch == '\r') { // 回车键
            password[index] = '\0';
            cout << endl;
            if (index < 6) {
                cout << userLanguage->passwordShort;
                index = 0; // 重置密码输入位置
            }
        }
        else {
            break;
        }
    }
}
```

```

    }
}
else {
    if (index < 20) { // 密码长度限制为20位
        cout << "*";
        password[index++] = ch;
    }
    else {
        cout << userLanguage->passwordLong;
        index = 0; // 重置密码输入位置
    }
}
}
string passwordStr(password);
MD5 md5;
return md5.calculate(passwordStr);
}

```

4. EmployeeProfile.h

```

#pragma once
#include <string>
#include "Language.h"
#include <memory>
using namespace std;
/**
 * @author XZH
 */
class EmployeeProfile
{
private:
    // 成员变量
    string id;           // 工号
    string name;         // 姓名
    string idNumber;     // 身份证号
    string gender;       // 性别
    int age;             // 年龄
    string phoneNumber;  // 联系电话
    string address;      // 家庭地址
    string education;    // 学历
    string position;     // 职位

```

```
string hireDate;        // 入职日期
string department;      // 所属部门
// EmployeeLanguage是一个智能指针，用于管理Language类的对象
static shared_ptr<Language> employeeLanguage;
// 通过第i项内容删除
static void deleteProfileByI(const string& filename, int i);
// 通过第i项内容查询并更新
static void updateProfileByI(const string& filename, int i);
public:
    // 无参构造函数
    EmployeeProfile() = default;
    // 有参构造函数
    EmployeeProfile(
        string id,
        string name,
        string idNumber,
        string gender,
        int age,
        string phoneNumber,
        string address,
        string education,
        string position,
        string hireDate,
        string department
    );
    // 用于初始化语言的构造函数
    EmployeeProfile(shared_ptr<Language>& langPtr);
    // 成员变量对应的get函数
    string getId();
    string getName();
    string getIdNumber();
    string getGender();
    int getAge();
    string getPhoneNumber();
    string getAddress();
    string getEducation();
    string getPosition();
    string getHireDate();
    string getDepartment();

    // 重载 >> 运算符
    friend istream& operator>>(istream& in, EmployeeProfile& profile);
```

```

// 重载 << 运算符
friend ostream& operator<<(ostream& out, EmployeeProfile& profile);
// 将信息存入文件
void saveEmployeeToFile(const string& filename);
// -----static-----
// 通过工号删除
static void deleteProfileById(const string& filename);
// 通过身份证号删除
static void deleteProfileByIdNumber(const string& filename);
// 根据工号更新职工
static void updateProfileById(const string& filename);
// 根据身份证号更新职工
static void updateProfileByIdNumber(const string& filename);
// 判断某个成员变量是否存在, i表示要查询的成员变量的序号
static bool isPropertyExists(const string& property, const string& filename,
const int i);
// 验证手机号是否正确
static bool verifyPhone(string phone);
};

```

5. EmployeeProfile.cpp

```

#include "EmployeeProfile.h"
#include <iostream>
#include <iomanip>
#include <fstream>
#include <conio.h>
#include <sstream>
#include <vector>
#include "Constant.h"
// #include <regex>
#include <ctime>
using namespace std;
shared_ptr<Language> EmployeeProfile::employeeLanguage = nullptr;
/**
 * @author XZH
 */
// 通过第i个属性删除职工
void EmployeeProfile::deleteProfileByI(const string& filename, int i)
{
    string id;

```

```
cin >> id;
ifstream inFile(filename);
if (!inFile.is_open()) {
    cerr << employeeLanguage->canNotOpen << endl;
    return;
}
// 创建临时文件来保存修改后的数据
string tempFilename = "temp_" + filename;
// 创建的就是新的空文件，不用ios_base::app追加形式
ofstream outFile(tempFilename);
if (!outFile.is_open()) {
    cerr << employeeLanguage->canCreateTemp << endl;
    inFile.close();
    return;
}
string line;
// 标记是否找到并删除了指定的记录
bool found = false;
while (getline(inFile, line)) {
    istringstream iss(line);
    string currentId;
    // 读取每行的第i个字段
    int tempI = i;
    while (tempI--) {
        getline(iss, currentId, ',');
    }

    if (currentId != id) {
        // 如果当前行的属性不是要删除的属性，则写入临时文件
        outFile << line << endl;
    }
    else {
        // 找到了匹配的工号
        found = true;
        // 此处进行break的话会导致要删除职工后面的职工没有写入新文件
    }
}
inFile.close();
outFile.close();
if (found) {
    // 如果找到了匹配的属性，删除原文件
    remove(filename.c_str());
}
```

```
// 将临时文件重命名为原文件名
rename(tempFilename.c_str(), filename.c_str());
if (i == EMPLOYEE_ID_SERIAL_NUMBER) {
    cout << employeeLanguage->employeeId << id <<
employeeLanguage->employeeAlreadyDel << endl;
}
else {
    cout << employeeLanguage->idNum << id <<
employeeLanguage->employeeAlreadyDel << endl;
}
}
else {
    // 如果没找到匹配项，删除临时文件
    remove(tempFilename.c_str());
    if (i == EMPLOYEE_ID_SERIAL_NUMBER) {
        cout << employeeLanguage->employeeIdNotFound << id <<
employeeLanguage->employeeRecord << endl;
    }
    else {
        cout << employeeLanguage->idNumNotFound << id <<
employeeLanguage->employeeRecord << endl;
    }
}
}
// 通过工号删除
void EmployeeProfile::deleteProfileById(const string& filename)
{
    cout << employeeLanguage->inputEmployeeId;
    deleteProfileByI(filename, EMPLOYEE_ID_SERIAL_NUMBER);
}
// 通过身份证号删除
void EmployeeProfile::deleteProfileByIdNumber(const string& filename)
{
    cout << employeeLanguage->inputIdNum;
    deleteProfileByI(filename, EMPLOYEE_ID_NUMBER_SERIAL_NUMBER);
}
// 将信息存入文件
void EmployeeProfile::saveEmployeeToFile(const string& filename)
{
    // ios_base::app为追加模式打开文件
    ofstream outFile(filename, ios_base::app);
    if (!outFile.is_open()) {
```



```
        cerr << employeeLanguage->canNotOpen << filename << endl;
        return;
    }
    // 在此再次判断, 如果工号存在则不保存到文件中
    if (EmployeeProfile::isPropertyExists(id, EMPLOYEE_FILENAME,
EMPLOYEE_ID_SERIAL_NUMBER)) {
        return;
    }
    // 在此再次判断, 如果身份证号存在则不保存到文件中
    if (EmployeeProfile::isPropertyExists(idNumber, EMPLOYEE_FILENAME,
EMPLOYEE_ID_NUMBER_SERIAL_NUMBER)) {
        return;
    }
    // 将职工信息写入文件, 中间用逗号分开, 便于之后解析
    outFile << id << "," << name << "," << idNumber << "," << gender << "," << age
<< ","
        << phoneNumber << "," << address << "," << education <<
        "," << position << "," << hireDate << "," << department << endl;
    outFile.close();
    cout << employeeLanguage->employeeSaveTo << filename << endl;
}
// 通过第i项内容查询并更新
void EmployeeProfile::updateProfileByI(const string& filename, int i)
{
    string id;
    cin >> id;
    ifstream inFile(filename);
    if (!inFile.is_open()) {
        cerr << employeeLanguage->canNotOpen << endl;
        return;
    }
    // 创建临时文件来保存修改后的数据
    string tempFilename = "temp_" + filename;
    // 创建的就是新的空文件, 不用ios_base::app追加形式
    ofstream outFile(tempFilename);
    if (!outFile.is_open()) {
        cerr << employeeLanguage->canCreateTemp << endl;
        inFile.close();
        return;
    }
    string line;
    // 标记是否找到并修改了指定的记录
```

```

bool found = false;
while (getline(inFile, line)) {
    istringstream iss(line);
    string currentId;
    // 读取每行的第i个字段（工号）
    int tempI = i;
    while (tempI--) {
        getline(iss, currentId, ',');
    }
    if (currentId != id) {
        // 如果当前行的属性不是要修改的属性，则写入临时文件
        outFile << line << endl;
    }
    else {
        // 找到了匹配的工号
        found = true;
        // 此处进行break的话会导致要删除职工后面的职工没有写入新文件

        // 使用vector来动态存储分割后的字符串
        vector<string> employeeVector;
        stringstream ss(line);
        string item;
        // 使用getline按逗号分割字符串，直到读取完所有元素
        while (getline(ss, item, ',')) {
            // 将分割得到的元素添加到vector中
            employeeVector.push_back(item);
        }
        string newE;
        for (int j = 0; j < employeeVector.size(); j++) {
            if (j == 0) {
                cout << employeeLanguage->retainOldVal << employeeVector[0] <<
endl
                << employeeLanguage->catNotUpdate << endl << endl;
                j++;
                // 清除输入缓冲区
                cin.ignore();
            }
            cout << employeeLanguage->oldVal << employeeVector[j] << endl <<
employeeLanguage->newVal;
            // 读取用户输入的字符
            char input = cin.get();
            // 如果不是回车键

```

```
if (input != '\n') {
    // 将读取的字符放回输入流
    cin.unget();
    // 进行输入操作
    cin >> newE;
    // 对身份证号进行校验
    if (j == EMPLOYEE_ID_NUMBER_SERIAL_NUMBER - 1) {
        while (1) {
            // 如果格式有误则重新输入
            if (!regex_match(newE, idNumberPattern)) {
                cout << employeeLanguage->idNumFormatError;
                cin >> newE;
                continue;
            }
            // 身份证号已存在
            if (EmployeeProfile::isPropertyExists(newE,
EMPLOYEE_FILENAME, EMPLOYEE_ID_NUMBER_SERIAL_NUMBER)) {
                cout <<employeeLanguage->idNumAlreadyExist;
                cin >> newE;
            }
            else {
                break;
            }
        }
    }
    // 对手机号进行校验
    if (j == EMPLOYEE_PHONE_SERIAL_NUMBER - 1) {
        while (1) {
            // 如果格式有误则重新输入
            if (!verifyPhone(newE)){
                cout << employeeLanguage->reinputPhone;
                cin >> newE;
                continue;
            }
            // 手机号号已存在
            if (EmployeeProfile::isPropertyExists(newE,
EMPLOYEE_FILENAME, EMPLOYEE_PHONE_SERIAL_NUMBER)) {
                cout << employeeLanguage->employeePhoneAlreadyExist;
                cin >> newE;
            }
            else {
                break;
            }
        }
    }
}
```

```
        }
    }
}

// 更新数据
employeeVector[j] = newE;
// 清除输入缓冲区
cin.ignore();
cout << endl;
}
else {
    cout << endl;
}
}

// 拼接新的职工信息
string newLine = "";
for (int j = 0; j < employeeVector.size(); j++) {
    // 如果不是最后一个则后面加逗号
    if (j != employeeVector.size() - 1) {
        newLine += employeeVector[j] + ",";
    }
    else {
        newLine += employeeVector[j];
    }
}

// 写入文件
outFile << newLine << endl;
}
}

inFile.close();
outFile.close();
if (found) {
    // 如果找到了匹配的属性，删除原文件
    remove(filename.c_str());
    // 将临时文件重命名为原文件名
    rename(tempFilename.c_str(), filename.c_str());
    if (i == EMPLOYEE_ID_SERIAL_NUMBER) {
        cout << employeeLanguage->employeeId << id <<
employeeLanguage->employeeUpdateSuccess << endl;
    }
    else {
        cout << employeeLanguage->idNum << id <<
employeeLanguage->employeeUpdateSuccess << endl;
    }
}
```

```

    }
}
else {
    // 如果没找到匹配项，删除临时文件
    remove(tempFilename.c_str());
    if (i == EMPLOYEE_ID_SERIAL_NUMBER) {
        cout << employeeLanguage->employeeIdNotFound << id <<
employeeLanguage->employeeRecord << endl;
    }
    else {
        cout << employeeLanguage->employeeIdNotFound << id <<
employeeLanguage->employeeRecord << endl;
    }
}
}
// 根据工号更新职工phonePattern
void EmployeeProfile::updateProfileById(const string& filename)
{
    cout << employeeLanguage->inputEmployeeId;
    updateProfileByI(filename, EMPLOYEE_ID_SERIAL_NUMBER);
}
// 根据身份证号更新职工
void EmployeeProfile::updateProfileByIdNumber(const string& filename)
{
    cout << employeeLanguage->inputIdNum;
    updateProfileByI(filename, EMPLOYEE_ID_NUMBER_SERIAL_NUMBER);
}
// 判断某个成员变量是否存在，i表示要查询的成员变量的序号
bool EmployeeProfile::isPropertyExists(const string& property, const string&
filename, const int i)
{
    ifstream inFile(filename);
    if (!inFile.is_open()) {
        cerr << employeeLanguage->canNotOpen << filename << endl;
        // 文件不存在或无法打开，视为用户名不存在
        return false;
    }
    string line;
    while (getline(inFile, line)) {
        istringstream iss(line);
        string currentProperty;
        // 读取每行的第i个字段

```

```
int tempI = i;
while (tempI--) {
    getline(iss, currentProperty, ',');
}
if (currentProperty == property && i == EMPLOYEE_ID_SERIAL_NUMBER) {
    inFile.close();
    return true;
}
else if (currentProperty == property && i ==
EMPLOYEE_ID_NUMBER_SERIAL_NUMBER) {
    inFile.close();
    return true;
}
else if (currentProperty == property && i == EMPLOYEE_PHONE_SERIAL_NUMBER) {
    inFile.close();
    return true;
}
}
inFile.close();
// 读取完所有行, 未找到匹配的工号
return false;
}
// 验证手机号是否正确
bool EmployeeProfile::verifyPhone(string phone) {
    // 如果手机号格式不正确
    if (!regex_match(phone, phonePattern)) {
        cout << employeeLanguage->phoneFormatError << endl;
        return false;
    }
}
// 重载 >> 运算符
istream& operator>>(istream& in, EmployeeProfile& profile) {
    while (1) {
        cout << EmployeeProfile::employeeLanguage->employeeIdFormat;
        in >> profile.id;
        // 如果格式有误则重新输入
        if (profile.id.size() == 10) {
            // 提取前四位
            string yearStr = profile.id.substr(0, 4);
            // 转换为int
            int yearInt = stoi(yearStr);
            // 获取当前时间
```

```
    time_t now = time(0);
    // 定义一个tm结构体变量来保存转换后的时间
    tm localTimeStruct;
    // 使用localtime_s进行安全的转换
    localtime_s(&localTimeStruct, &now);
    // 获取年份, tm_year是从1900年开始的, 所以需要加1900
    int currentYear = localTimeStruct.tm_year + 1900;
    if (yearInt < 1956 || yearInt > currentYear) {
        cout << EmployeeProfile::employeeLanguage->employeeIdFormatError <<
endl;
        continue;
    }
}
else {
    cout << EmployeeProfile::employeeLanguage->employeeIdFormatError << endl;
    continue;
}
// 工号已存在
if (EmployeeProfile::isPropertyExists(profile.id, EMPLOYEE_FILENAME,
EMPLOYEE_ID_SERIAL_NUMBER)) {
    cout << EmployeeProfile::employeeLanguage->employeeIdAlreadyExist <<
endl;
}
else {
    break;
}
}
cout << EmployeeProfile::employeeLanguage->employeeName;
in >> profile.name;
while (1) {
    cout << EmployeeProfile::employeeLanguage->inputIdNum2;
    in >> profile.idNumber;
    // 如果格式有误则重新输入
    if (!regex_match(profile.idNumber, idNumberPattern)) {
        cout << EmployeeProfile::employeeLanguage->idNumFormatError2 << endl;
        continue;
    }
    // 身份证号已存在
    if (EmployeeProfile::isPropertyExists(profile.idNumber, EMPLOYEE_FILENAME,
EMPLOYEE_ID_NUMBER_SERIAL_NUMBER)) {
        cout << EmployeeProfile::employeeLanguage->idNumAlreadyExist2 << endl;
    }
}
```

```

        else {
            break;
        }
    }
    cout << EmployeeProfile::employeeLanguage->gender;
    in >> profile.gender;
    cout << EmployeeProfile::employeeLanguage->age;
    in >> profile.age;
    while (1) {
        cout << EmployeeProfile::employeeLanguage->employeePhone;
        in >> profile.phoneNumber;
        // 如果格式有误则重新输入
        if (!EmployeeProfile::verifyPhone(profile.phoneNumber)) {
            continue;
        }
        // 身份证号已存在
        if (EmployeeProfile::isPropertyExists(profile.phoneNumber,
EMPLOYEE_FILENAME, EMPLOYEE_PHONE_SERIAL_NUMBER)) {
            cout << EmployeeProfile::employeeLanguage->phoneAlreadyExist2 << endl;
        }
        else {
            break;
        }
    }

    cout << EmployeeProfile::employeeLanguage->address;
    in >> profile.address;
    cout << EmployeeProfile::employeeLanguage->education;
    in >> profile.education;
    cout << EmployeeProfile::employeeLanguage->position;
    in >> profile.position;
    cout << EmployeeProfile::employeeLanguage->hireDate;
    in >> profile.hireDate;
    cout << EmployeeProfile::employeeLanguage->department;
    in >> profile.department;
    return in;
}

// 重载 << 运算符
ostream& operator<<(ostream& out, EmployeeProfile& profile)
{
    const int SHORT_WIDTH = 10; // 短列宽
    const int MIDDLE_WIDTH = 15; // 中列宽

```



```
const int LONG_WIDTH = 20; // 长列宽
const int LINE_LENGTH = 155; // 分割线长度
// 输出数据行
out << left << setw(MIDDLE_WIDTH) << profile.id
    << setw(SHORT_WIDTH) << profile.name
    << setw(LONG_WIDTH) << profile.idNumber
    << setw(SHORT_WIDTH) << profile.gender
    << setw(SHORT_WIDTH) << profile.age
    << setw(MIDDLE_WIDTH) << profile.phoneNumber
    << setw(LONG_WIDTH) << profile.address
    << setw(SHORT_WIDTH) << profile.education
    << setw(MIDDLE_WIDTH) << profile.position
    << setw(MIDDLE_WIDTH) << profile.hireDate
    << setw(MIDDLE_WIDTH) << profile.department << endl;
out << string(LINE_LENGTH, '-');
return out;
}
// 有参构造函数定义
EmployeeProfile::EmployeeProfile(
    string i,
    string n,
    string idN,
    string g,
    int a,
    string p,
    string addr,
    string e,
    string pos,
    string hd,
    string dep
) :
    id(i),
    name(n),
    idNumber(idN),
    gender(g),
    age(a),
    phoneNumber(p),
    address(addr),
    education(e),
    position(pos),
    hireDate(hd),
    department(dep)
```

```
{
}
EmployeeProfile::EmployeeProfile(shared_ptr<Language>& langPtr) {
    employeeLanguage = langPtr;
}
// 成员变量对应的get函数
string EmployeeProfile::getId()
{
    return id;
}
string EmployeeProfile::getName()
{
    return name;
}
string EmployeeProfile::getIdNumber()
{
    return idNumber;
}
string EmployeeProfile::getGender()
{
    return gender;
}
int EmployeeProfile::getAge()
{
    return age;
}
string EmployeeProfile::getPhoneNumber()
{
    return phoneNumber;
}
string EmployeeProfile::getAddress()
{
    return address;
}
string EmployeeProfile::getEducation()
{
    return education;
}
string EmployeeProfile::getPosition()
{
    return position;
}
```

```
string EmployeeProfile::getHireDate()
{
    return hireDate;
}
string EmployeeProfile::getDepartment()
{
    return department;
}
```

6. Language.h

```
#pragma once
#include <string>
using namespace std;
class Language
{
public:
    virtual ~Language() {}
    // 登录注册页面显示
    virtual string loginAndRegisterPage() const = 0;
    virtual string menu() const = 0;
    virtual string tableTitle() const = 0;
    string welcomeLogin;
    string welcomeRegister;
    string forgetPassword;
    string isExit;
    string yesOrNo;
    string selectFunction;
    string currentPage;
    string page;
    string leftOrRight;
    string totalEmployee;
    string updateByIdOrIdNumber;
    string deleteByIdOrIdNumber;
    string selectTime;
    string close;
    string userName;
    string music;
    string yesOrNo2;
    string exitSuccess;
    string loginSuccess;
```

```
string registerSuccess;
string inputUserName;
string inputConfirmPassword;
string inputPhone;
string inputPassword;
string inputCode;
string inputPageNum;
string inputNum;
string inputNumPerPage;
string inputOneOrTwo;
string inputSelectContent;
string userNameNotExist;
string phoneAlreadyExist;
string userNameAlreadyExist;
string passwordDifferent;
string userNameOrPasswordError;
string overtime;
string codeError;
string numTooLong;
string canNotOpen;
string pressEnter;
string KEY_ERROR;

// -----User-----
string canCreateTemp;
string inputNewPassword;
string phone;
string updatePasswordSuccess;
string saveUser;
string wait;
string sending;
string sended;
string sendCode;
string timeValid;
string phoneNotRegister;
string passwordShort;
string passwordLong;
// -----EmployeeProfile-----
string employeeId;
string employeeAlreadyDel;
string idNum;
string employeeIdNotFound;
```

```
string employeeRecord;
string idNumNotFound;
string inputEmployeeId;
string inputIdNum;
string employeeSaveTo;
string rатаinOldVal;
string catNotUpdate;
string oldVal;
string newVal;
string idNumFormatError;
string idNumAlreadyExist;
string reinputPhone;
string employeePhoneAlreadyExist = "手机号已存在！\n请重新输入手机号：";
string employeeUpdateSuccess;
string phoneFormatError;
string employeeIdFormat;
string employeeIdFormatError;
string employeeIdAlreadyExist;
string employeeName;
string inputIdNum2;
string idNumFormatError2;
string idNumAlreadyExist2;
string gender;
string age;
string employeePhone;
string phoneAlreadyExist2;
string address;
string education;
string position;
string hireDate;
string department;
};
```

7. Chinese.h

```
#pragma once
#include "Language.h"

using namespace std;
class Chinese : public Language
{
```

```

public:
    string loginAndRegisterPage() const override;
    string menu() const override;
    string tableTitle() const override;
    Chinese();
};

```

8. Chinese.cpp

```

#include "Chinese.h"
#include <string>
#include <iostream>
#include <iomanip>
using namespace std;

string Chinese::loginAndRegisterPage() const {
    cout << "    欢迎使用职工档案管理系统    " << endl;
    cout << "*****" << endl;
    cout << "*****          *****" << endl;
    cout << "*****    1. 登录          *****" << endl;
    cout << "*****          *****" << endl;
    cout << "*****    2. 注册          *****" << endl;
    cout << "*****          *****" << endl;
    cout << "*****    3. 忘记密码      *****" << endl;
    cout << "*****          *****" << endl;
    cout << "*****    ESC.退出系统      *****" << endl;
    cout << "*****          *****" << endl;
    cout << "*****" << endl;
    return "";
}

string Chinese::menu() const
{
    cout << "    欢迎使用职工档案管理系统    " << endl;
    cout << "_____ " << endl;
    cout << "|                                |" << endl;
    cout << "|    1. 查询职工档案信息        |" << endl;
    cout << "|    2. 新增职工档案信息        |" << endl;
    cout << "|    3. 修改职工档案信息        |" << endl;
    cout << "|    4. 删除职工档案信息        |" << endl;
    cout << "|    ESC.退出登录              |" << endl;
    cout << "|                                |" << endl;
    cout << "_____ " << endl;
}

```

```

        return "";
    }
    string Chinese::tableTitle() const
    {
        const int SHORT_WIDTH = 10; // 短列宽
        const int MIDDLE_WIDTH = 15; // 中列宽
        const int LONG_WIDTH = 20; // 长列宽
        const int LINE_LENGTH = 155; // 分割线长度
        // 输出表头
        cout << left
            << setw(MIDDLE_WIDTH) << "工号:"
            << setw(SHORT_WIDTH) << "职工姓名:"
            << setw(LONG_WIDTH) << "身份证号:"
            << setw(SHORT_WIDTH) << "性别:"
            << setw(SHORT_WIDTH) << "年龄:"
            << setw(MIDDLE_WIDTH) << "联系电话:"
            << setw(LONG_WIDTH) << "家庭地址:"
            << setw(SHORT_WIDTH) << "学历:"
            << setw(MIDDLE_WIDTH) << "职位:"
            << setw(MIDDLE_WIDTH) << "入职日期:"
            << setw(MIDDLE_WIDTH) << "所属部门:"
            << endl;
        return "";
    }
    Chinese::Chinese()
    {
        welcomeLogin = "  欢迎登录职工档案管理系统  ";
        welcomeRegister = "  欢迎注册职工档案管理系统  ";
        forgetPassword = "  忘记密码  ";
        isExit = "是否退出";
        yesOrNo = "1.确认2.取消";
        selectFunction = "\t    1.按工号升序 2.按工号降序 3.按年龄升序 4.按年龄降序"
            "5.按入职时间升序 6.按入职时间降序 7.前往页码 8.更改每页展示数 9.模糊查询 ESC键退"
            "出";
        currentPage = "\n当前页: ";
        page = "/页";
        leftOrRight = "\t向左: A/←\t向右: D/→";
        totalEmployee = "总职工数: ";
        updateByIdOrIdNumber = "1.通过工号更新职工 2.通过身份证号更新职工 ESC键退出";
        deleteByIdOrIdNumber = "1.通过工号删除 2.通过身份证号删除 ESC键退出";
        selectTime = "本次查询时间: ";
        close = "3秒后关闭窗口.";
    }

```

```
userName = "用户名: ";
music = "是否开启背景音乐? ";
yesOrNo2 = "1.是 2.否";
exitSuccess = "\n退出成功! ";
loginSuccess = "登陆成功";
registerSuccess = "注册成功";
inputUserName = "请输入用户名: ";
inputPassword = "请输入密码: ";
inputConfirmPassword = "请再次输入密码: ";
inputPhone = "请输入手机号: ";
inputCode = "请输入验证码: ";
inputPageNum = "\n请输入页码: ";
inputNum = "请输入数字! ";
inputNumPerPage = "\n请输入每页展示数: ";
inputOneOrTwo = "请按1或2键";
inputSelectContent = "\n请输入查询的内容: ";
userNameNotExist = "用户名不存在, 请先注册";
userNameAlreadyExist = "用户名已存在, 请重新输入";
phoneAlreadyExist = "手机号已存在, 请重新输入";
passwordDifferent = "两次密码不一致, 请重新输入! ";
userNameOrPasswordError = "用户名或密码错误, 请重试! ";
overtime = "超过60秒, 操作超时";
codeError = "验证码错误, 请重新输入: ";
numTooLong = "数字太大, 超出了允许的范围! ";
canNotOpen = "无法打开文件: ";
pressEnter = "按回车键重新发送";
KEY_ERROR = "请按正确的按键! ";
// -----User-----
canCreateTemp = "无法创建临时文件! ";
inputNewPassword = "请输入新密码: ";
phone = "手机号为 ";
updatePasswordSuccess = " 的用户密码已成功修改.";
saveUser = "用户数据已保存至: ";
wait = "请稍等...";
sending = "正在发送";
sended = "已向手机号 ";
sendCode = " 发送验证码: ";
timeValid = "\t(60秒有效)";
phoneNotRegister = "该手机号未注册! ";
passwordShort = "密码长度不能少于6位, 请重新输入: ";
passwordLong = "\n密码长度不能超过20位, 请重新输入: ";
// -----EmployeeProfile-----
```



```

employeeId = "工号为 ";
employeeAlreadyDel = " 的员工记录已成功删除。";
idNum = "身份证为 ";
employeeIdNotFound = "未找到工号为 ";
employeeRecord = " 的员工记录。";
idNumNotFound = "未找到身份证号为 ";
inputEmployeeId = "请输入工号：";
inputIdNum = "请输入身份证号：";
employeeSaveTo = "职工数据已保存至：";
retainOldVal = "按回车键则保留旧数据\n\n旧值：";
catNotUpdate = "不可修改";
oldVal = "旧值：";
newVal = "新值：";
idNumFormatError = "身份证号格式有误！\n请重新输入身份证号：";
idNumAlreadyExist = "身份证号已存在！\n请重新输入身份证号：";
reinputPhone = "请重新输入手机号：";
employeePhoneAlreadyExist = "手机号已存在！\n请重新输入手机号：";
employeeUpdateSuccess = " 的员工记录已成功修改。";
phoneFormatError = "手机号格式错误！";
employeeIdFormat = "工号(长度为10位,前四位数为1956~当前年份之间)";
employeeIdFormatError = "工号格式有误！";
employeeIdAlreadyExist = "工号已存在！";
employeeName = "职工姓名:";
inputIdNum2 = "身份证号（X为大写）:";
idNumFormatError2 = "身份证号格式有误！";
idNumAlreadyExist2 = "身份证号已存在！";
gender = "性别:";
age = "年龄:";
employeePhone = "联系电话:";
phoneAlreadyExist2 = "手机号已存在！";
address = "家庭地址:";
education = "学历:";
position = "职位:";
hireDate = "入职日期:";
department = "所属部门:";
}

```

9. English.h

```

#pragma once
#include "Language.h"

```

```

#include <string>
using namespace std;
class English : public Language
{
public:
    string loginAndRegisterPage() const override;
    string menu() const override;
    string tableTitle() const override;
    English();
};

```

10. English.cpp

```

#include "English.h"
#include <string>
#include <iostream>
#include <iomanip>
using namespace std;
string English::loginAndRegisterPage() const {
    cout << "Welcome to use the employee file management system" << endl;
    cout << " *****" << endl;
    cout << " ****          *****" << endl;
    cout << " ****      1. login          *****" << endl;
    cout << " ****          *****" << endl;
    cout << " ****      2. register        *****" << endl;
    cout << " ****          *****" << endl;
    cout << " ****      3. forget password *****" << endl;
    cout << " ****          *****" << endl;
    cout << " ****      ESC.exit system    *****" << endl;
    cout << " ****          *****" << endl;
    cout << " *****" << endl;
    return "";
}
string English::menu() const
{
    cout << "      Welcome to the Employee Profile Management System  " << endl;
    cout << " _____" << endl;
    cout << " |                                     |" << endl;
    cout << " |      1. Query Employee Profiles  |" << endl;
    cout << " |      2. Add Employee Profile     |" << endl;
}

```

```

    cout << "|      3. Update Employee Profile      |" << endl;
    cout << "|      4. Delete Employee Profile      |" << endl;
    cout << "|      ESC. Logout                        |" << endl;
    cout << "|                                          |" << endl;
    cout << "_____ " << endl;
    return "";
}

string English::tableTitle() const
{
    const int SHORT_WIDTH = 10; // 短列宽
    const int MIDDLE_WIDTH = 15; // 中列宽
    const int LONG_WIDTH = 20; // 长列宽
    const int LINE_LENGTH = 155; // 分割线长度
    // 输出表头
    cout << left
        << setw(MIDDLE_WIDTH) << "Employee ID:"
        << setw(SHORT_WIDTH) << "Name:"
        << setw(LONG_WIDTH) << "ID Number:"
        << setw(SHORT_WIDTH) << "Gender:"
        << setw(SHORT_WIDTH) << "Age:"
        << setw(MIDDLE_WIDTH) << "Phone:"
        << setw(LONG_WIDTH) << "Address:"
        << setw(SHORT_WIDTH) << "Education:"
        << setw(MIDDLE_WIDTH) << "Position:"
        << setw(MIDDLE_WIDTH) << "Hire Date:"
        << setw(MIDDLE_WIDTH) << "Department:"
        << endl;
    return "";
}

English::English()
{
    welcomeLogin = " Welcome to Login the Employee Profile Management System ";
    welcomeRegister = " Welcome to Register in the Employee Profile Management
System ";
    forgetPassword = " Forgot Password ";
    isExit = "Exit?";
    yesOrNo = "1. Yes 2. No";
    selectFunction = "\t 1. Ascending by Employee ID 2. Descending by Employee ID
3. Ascending by Age 4. Descending by Age"
        "\t5. Ascending by Hire Date \n\t 6. Descending by Hire Date 7. Go to Page
8. Change Items Per Page 9. Fuzzy Query \t\tESC to Exit";
    currentPage = "\nCurrent Page: ";
}

```

```

page = "/Page";
leftOrRight = "\tLeft: A/←\tRight: D/→";
totalEmployee = "Total Employees: ";
updateByIdOrIdNumber = "1. Update Employee by ID 2. Update Employee by ID
Number Press \t\tESC to Exit";
deleteByIdOrIdNumber = "1. Delete Employee by ID 2. Delete Employee by ID
Number Press \t\tESC to Exit";
selectTime = "Current Query Time: ";
close = "Window will close in 3 seconds.";
userName = "Username: ";
music = "Whether to enable background music?";
yesOrNo2 = "1.YES 2.NO";
exitSuccess = "\nExit successful!";
loginSuccess = "Login successful";
registerSuccess = "Registration successful";
inputUserName = "Please enter username: ";
inputPassword = "Please enter password: ";
inputConfirmPassword = "Please enter password again: ";
inputPhone = "Please enter phone number: ";
inputCode = "Please enter verification code: ";
inputPageNum = "\nPlease enter page number: ";
inputNum = "Please enter a number!";
inputNumPerPage = "\nPlease enter items per page: ";
inputOneOrTwo = "Please press 1 or 2";
inputSelectContent = "\nPlease enter query content: ";
userNameNotExist = "Username does not exist, please register first";
userNameAlreadyExist = "Username already exists, please enter a different one";
phoneAlreadyExist = "Phone number already exists, please enter a different
one";
passwordDifferent = "Passwords do not match, please try again!";
userNameOrPasswordError = "Incorrect username or password, please try again!";
overtime = "Operation timed out, over 60 seconds";
codeError = "Incorrect verification code, please try again: ";
numTooLong = "Number is too large, exceeds the allowed range!";
canNotOpen = "Cannot open file: ";
pressEnter = "Press Enter to resend";
KEY_ERROR = "Please press the correct KEY!";
// -----User-----
canCreateTemp = "Cannot create temporary file!";
inputNewPassword = "Please enter new password: ";
phone = "Phone number is ";
updatePasswordSuccess = " user's password has been successfully updated.";

```

```

saveUser = "User data saved to: ";
wait = "Please wait...";
sending = "Sending";
sended = "Code has been sent to phone number ";
sendCode = " Send code: ";
timeValid = "\t(Valid for 60 seconds)";
phoneNotRegister = "This phone number is not registered!";
passwordShort = "Password length should be at least 6 characters, please try
again: ";
passwordLong = "\nPassword length should not exceed 20 characters, please try
again: ";
// -----EmployeeProfile-----
employeeId = "Employee ID is ";
employeeAlreadyDel = " employee record has been successfully deleted.";
idNum = "ID number is ";
employeeIdNotFound = "Employee record with ID ";
employeeRecord = " not found.";
idNumNotFound = "Employee record with ID number ";
inputEmployeeId = "Please enter employee ID: ";
inputIdNum = "Please enter ID number: ";
employeeSaveTo = "Employee data saved to: ";
retainOldVal = "Press Enter to retain old data\n\nOld value: ";
catNotUpdate = "Cannot update";
oldVal = "Old value: ";
newVal = "New value: ";
idNumFormatError = "Invalid ID number format!\nPlease enter ID number again: ";
idNumAlreadyExist = "ID number already exists!\nPlease enter ID number again:
";
reinputPhone = "Please re-enter phone number: ";
employeePhoneAlreadyExist = "Phone number already exists!\nPlease enter phone
number again: ";
employeeUpdateSuccess = " employee record has been successfully updated.";
phoneFormatError = "Invalid phone number format!";
employeeIdFormat = "Employee ID (length is 10 digits, first four digits are
between 1956 and the current year):";
employeeIdFormatError = "Invalid employee ID format!";
employeeIdAlreadyExist = "Employee ID already exists!";
employeeName = "Employee name: ";
inputIdNum2 = "ID number (X is uppercase): ";
idNumFormatError2 = "Invalid ID number format!";
idNumAlreadyExist2 = "ID number already exists!";
gender = "Gender: ";

```

```

    age = "Age: ";
    employeePhone = "Phone: ";
    phoneAlreadyExist2 = "Phone number already exists!";
    address = "Address: ";
    education = "Education: ";
    position = "Position: ";
    hireDate = "Hire Date: ";
    department = "Department: ";
}

```

11. Constant.h

```

#pragma once
#include <string>
#include <regex>
using namespace std;
/**
 * @author XZH
 */
const string EMPLOYEE_FILENAME = "employee.txt";
const string USER_FILENAME = "user.txt";
const string KEY_ERROR = "请按正确的按键! ";
const string KEY_ERROR_ENG = "Please press the correct KEY";
const int PAGE_NUM = 1;
const int PAGE_SIZE = 6;
static int pageNum = 1;
static int pageSize = 6;
// 序号为该属性在一行中的位置, 从 1 开始
// 职工 id 的序号
static int EMPLOYEE_ID_SERIAL_NUMBER = 1;
// 职工 phone 的序号
static int EMPLOYEE_PHONE_SERIAL_NUMBER = 6;
// 职工 idNumber 的序号
static int EMPLOYEE_ID_NUMBER_SERIAL_NUMBER = 3;
// 用户 phone 的序号
static int USER_PHONE_SERIAL_NUMBER = 3;
// 用户 password 的序号
static int USER_PASSWORD_SERIAL_NUMBER = 2;
// 用户 name 的序号
static int USER_NAME_SERIAL_NUMBER = 1;
// 身份证正则表达式

```

```
static regex idNumberPattern("^([1-9]\\d{5}(?:18|19|20)\\d{2}(?:0\\d|10|11|12)(?:0[1-9]|[1-2]\\d|30|31)\\d{3}[\\dX]$");
// 手机号正则表达式
static regex phonePattern("^(?:0086)?1(?:3[\\d])|(?:4[5-7|9])|(?:5[0-3|5-9])|(?:6[5-7])|(?:7[0-8])|(?:8[\\d])|(?:9[1|8|9]))\\d{8}$");
```

12. MD5.h

```
#pragma once
#include <iostream>
#include <string>
#include <cstring>
#include <cmath>
/**
 * @author XZH
 */
// Functions for MD5 transformation
#define F(x, y, z) (((x) & (y)) | ((~x) & (z)))
#define G(x, y, z) (((x) & (z)) | ((y) & (~z)))
#define H(x, y, z) ((x) ^ (y) ^ (z))
#define I(x, y, z) ((y) ^ ((x) | (~z)))
#define ROTATE_LEFT(x, n) (((x) << (n)) | ((x) >> (32-(n))))
#define FF(a, b, c, d, x, s, ac) { \
    (a) += F((b), (c), (d)) + (x) + (ac); \
    (a) = ROTATE_LEFT((a), (s)); \
    (a) += (b); \
}
#define GG(a, b, c, d, x, s, ac) { \
    (a) += G((b), (c), (d)) + (x) + (ac); \
    (a) = ROTATE_LEFT((a), (s)); \
    (a) += (b); \
}
#define HH(a, b, c, d, x, s, ac) { \
    (a) += H((b), (c), (d)) + (x) + (ac); \
    (a) = ROTATE_LEFT((a), (s)); \
    (a) += (b); \
}
#define II(a, b, c, d, x, s, ac) { \
    (a) += I((b), (c), (d)) + (x) + (ac); \
    (a) = ROTATE_LEFT((a), (s)); \
}
```

```

        (a) += (b); \
    }
class MD5 {
public:
    MD5() {
        init();
    }
    std::string calculate(const std::string& input) {
        init();
        update(reinterpret_cast<const unsigned char*>(input.c_str()),
input.length());
        finalize();
        char buf[33];
        for (int i = 0; i < 16; i++) {
            sprintf_s(&buf[i * 2], 3, "%02x", static_cast<unsigned int>(digest[i]));
        }
        buf[32] = '\0';
        return std::string(buf);
    }
private:
    void init() {
        count[0] = count[1] = 0;
        state[0] = 0x67452301;
        state[1] = 0xefcdab89;
        state[2] = 0x98badcfe;
        state[3] = 0x10325476;
    }
    void update(const unsigned char* input, size_t input_len) {
        size_t i, index, part_len;
        index = (count[0] >> 3) & 0x3F;
        if ((count[0] += (input_len << 3)) < (input_len << 3))
            count[1]++;
        count[1] += (input_len >> 29);
        part_len = 64 - index;
        if (input_len >= part_len) {
            memcpy(&buffer[index], input, part_len);
            transform(buffer);
            for (i = part_len; i + 63 < input_len; i += 64) {
                transform(&input[i]);
            }
            index = 0;
        }
    }
}

```



```
else {
    i = 0;
}
memcpy(&buffer[index], &input[i], input_len - i);
}

void finalize() {
    unsigned char bits[8];
    size_t index, pad_len;
    encode(bits, count, 8);
    index = (count[0] >> 3) & 0x3f;
    pad_len = (index < 56) ? (56 - index) : (120 - index);
    update(padding, pad_len);
    update(bits, 8);
    encode(digest, state, 16);
}

void transform(const unsigned char block[64]) {
    uint32_t a = state[0], b = state[1], c = state[2], d = state[3], x[16];
    decode(x, block, 64);
    /* Round 1 */
    FF(a, b, c, d, x[0], 7, 0xd76aa478);
    FF(d, a, b, c, x[1], 12, 0xe8c7b756);
    FF(c, d, a, b, x[2], 17, 0x242070db);
    FF(b, c, d, a, x[3], 22, 0xc1bdceee);
    FF(a, b, c, d, x[4], 7, 0xf57c0faf);
    FF(d, a, b, c, x[5], 12, 0x4787c62a);
    FF(c, d, a, b, x[6], 17, 0xa8304613);
    FF(b, c, d, a, x[7], 22, 0xfd469501);
    FF(a, b, c, d, x[8], 7, 0x698098d8);
    FF(d, a, b, c, x[9], 12, 0x8b44f7af);
    FF(c, d, a, b, x[10], 17, 0xffff5bb1);
    FF(b, c, d, a, x[11], 22, 0x895cd7be);
    FF(a, b, c, d, x[12], 7, 0x6b901122);
    FF(d, a, b, c, x[13], 12, 0xfd987193);
    FF(c, d, a, b, x[14], 17, 0xa679438e);
    FF(b, c, d, a, x[15], 22, 0x49b40821);
    /* Round 2 */
    GG(a, b, c, d, x[1], 5, 0xf61e2562);
    GG(d, a, b, c, x[6], 9, 0xc040b340);
    GG(c, d, a, b, x[11], 14, 0x265e5a51);
    GG(b, c, d, a, x[0], 20, 0xe9b6c7aa);
    GG(a, b, c, d, x[5], 5, 0xd62f105d);
    GG(d, a, b, c, x[10], 9, 0x02441453);
```

```
GG(c, d, a, b, x[15], 14, 0xd8a1e681);
GG(b, c, d, a, x[4], 20, 0xe7d3fbc8);
GG(a, b, c, d, x[9], 5, 0x21e1cde6);
GG(d, a, b, c, x[14], 9, 0xc33707d6);
GG(c, d, a, b, x[3], 14, 0xf4d50d87);
GG(b, c, d, a, x[8], 20, 0x455a14ed);
GG(a, b, c, d, x[13], 5, 0xa9e3e905);
GG(d, a, b, c, x[2], 9, 0xfcefa3f8);
GG(c, d, a, b, x[7], 14, 0x676f02d9);
GG(b, c, d, a, x[12], 20, 0x8d2a4c8a);
/* Round 3 */
HH(a, b, c, d, x[5], 4, 0xfffa3942);
HH(d, a, b, c, x[8], 11, 0x8771f681);
HH(c, d, a, b, x[11], 16, 0x6d9d6122);
HH(b, c, d, a, x[14], 23, 0xfde5380c);
HH(a, b, c, d, x[1], 4, 0xa4beea44);
HH(d, a, b, c, x[4], 11, 0x4bdecfa9);
HH(c, d, a, b, x[7], 16, 0xf6bb4b60);
HH(b, c, d, a, x[10], 23, 0xbebfbcb70);
HH(a, b, c, d, x[13], 4, 0x289b7ec6);
HH(d, a, b, c, x[0], 11, 0xeeaa127fa);
HH(c, d, a, b, x[3], 16, 0xd4ef3085);
HH(b, c, d, a, x[6], 23, 0x04881d05);
HH(a, b, c, d, x[9], 4, 0xd9d4d039);
HH(d, a, b, c, x[12], 11, 0xe6db99e5);
HH(c, d, a, b, x[15], 16, 0x1fa27cf8);
HH(b, c, d, a, x[2], 23, 0xc4ac5665);
/* Round 4 */
II(a, b, c, d, x[0], 6, 0xf4292244);
II(d, a, b, c, x[7], 10, 0x432aff97);
II(c, d, a, b, x[14], 15, 0xab9423a7);
II(b, c, d, a, x[5], 21, 0xfc93a039);
II(a, b, c, d, x[12], 6, 0x655b59c3);
II(d, a, b, c, x[3], 10, 0x8f0ccc92);
II(c, d, a, b, x[10], 15, 0xffeff47d);
II(b, c, d, a, x[1], 21, 0x85845dd1);
II(a, b, c, d, x[8], 6, 0x6fa87e4f);
II(d, a, b, c, x[15], 10, 0xfe2ce6e0);
II(c, d, a, b, x[6], 15, 0xa3014314);
II(b, c, d, a, x[13], 21, 0x4e0811a1);
II(a, b, c, d, x[4], 6, 0xf7537e82);
II(d, a, b, c, x[11], 10, 0xbd3af235);
```

```

    II(c, d, a, b, x[2], 15, 0x2ad7d2bb);
    II(b, c, d, a, x[9], 21, 0xeb86d391);
    state[0] += a;
    state[1] += b;
    state[2] += c;
    state[3] += d;
    memset(x, 0, sizeof(x));
}

void encode(unsigned char* output, const uint32_t* input, size_t len) {
    size_t i, j;
    for (i = 0, j = 0; j < len; i++, j += 4) {
        output[j] = static_cast<unsigned char>(input[i] & 0xff);
        output[j + 1] = static_cast<unsigned char>((input[i] >> 8) & 0xff);
        output[j + 2] = static_cast<unsigned char>((input[i] >> 16) & 0xff);
        output[j + 3] = static_cast<unsigned char>((input[i] >> 24) & 0xff);
    }
}

void decode(uint32_t* output, const unsigned char* input, size_t len) {
    size_t i, j;
    for (i = 0, j = 0; j < len; i++, j += 4) {
        output[i] = (static_cast<uint32_t>(input[j])) |
            (static_cast<uint32_t>(input[j + 1]) << 8) |
            (static_cast<uint32_t>(input[j + 2]) << 16) |
            (static_cast<uint32_t>(input[j + 3]) << 24);
    }
}

uint32_t state[4];
uint32_t count[2];
unsigned char buffer[64];
unsigned char digest[16];
static const unsigned char padding[64];
};

const unsigned char MD5::padding[64] = {
    0x80, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
};

```

东华理工大学

课程设计评分表

学生姓名：

班级：

学号：

课程设计题目：

	项目内容	满分	实 评
选题	能结合所学课程知识、有一定的能力训练。符合选题要求（5 人一题）	10	
	工作量适中，难易度合理	10	
能力水平	能熟练应用所学知识，有一定查阅文献及运用文献资料能力	10	
	理论依据充分，数据准确，公式推导正确	10	
	能应用计算机软件进行编程、资料搜集录入、加工、排版、制图等	10	
	能体现创造性思维，或有独特见解	10	
成果质量	总体设计正确、合理，各项技术指标符合要求。	10	
	说明书综述简练完整，概念清楚、立论正确、技术用语准确、结论严谨合理；分析处理科学、条理分明、语言流畅、结构严谨、版面清晰	10	
	设计说明书栏目齐全、合理，符号统一、编号齐全。格式、绘图、表格、插图等规范准确，符合国家标准	10	
	有一定篇幅，字符数不少于 5000	10	
	总 分	100	

指导教师评语：

指导教师签名：

年 月 日