浙江工业大学

C++程序设计
课程设计报告

2022/2023(2)

实验题目 公司员工管理系统

学生姓名 陈王子

学生学号 202103150503

学生班级 大数据分析 2101 班

任课教师 毛国红

提交日期 2023 年 6 月 3 日

计算机科学与技术学院

公司员工管理系统 实验报告

一、 实验内容

公司员工管理系统（Employee Manage System, EMS）用于企业进行员工信息的管理。每个员工的信息包括编号、姓名、年龄、受教育程度、民族、专业、职称（助理工程师、工程师、高级工程师、教授级高级工程师）、部门（人事部、技术部、后勤部、信息部）、职务（普通员工、工程师、团队领导、部门领导、公司领导）。

公司员工管理系统完成了以下要求：

1、良好的人际交互界面、方便操作；设计菜单实现功能选择；

2、所有数据均保存到文件，并可以从文件中读出；

3、提供增加、修改、查询、删除员工数据信息的功能；

4、显示各职称的员工信息、显示各部门的员工数量；

5、能够根据不同的字段信息排序显示；

6、所有的增加、修改、删除能同步到文件；也从文件读取数据到程序。

7、考虑各项数据的合法性检查

二、 运行环境

开发环境：Microsoft Visual Studio Community 2019 版本 16.11.7

操作系统：Windows 10.0.19045 内部版本 19045

处理器：11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz (16CPUs)

内存：32.00GB

系统类型：64 位操作系统

三、 实验课题分析

3.1 员工管理系统的主要功能

员工管理系统（EMS）主要功能为：添加员工、修改员工信息、删除员工、根据员工 ID 查询员工信息、统计各职称员工信息、统计各部门的员工数量、根据不同的字段信息排序显示、显示所有员工信息、从文件读取员工信息、向文件写入员工信息。

详细的系统功能结构为图 1 所示。

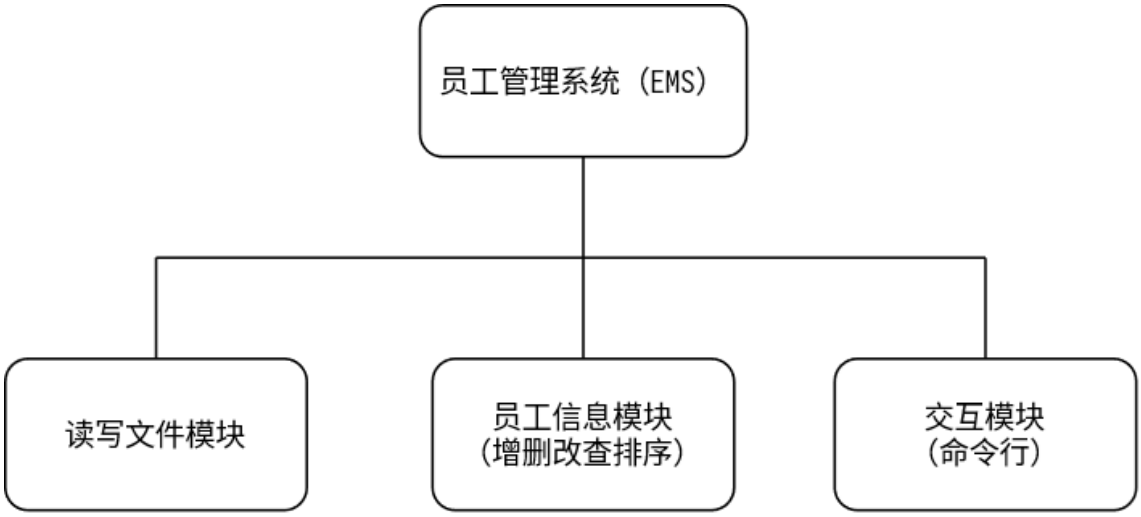


图 1 *系统结构图*

系统各模块的功能具体描述为：

1. 读写文件模块：

该模块主要负责从文件中读取员工信息或将员工信息写入文件。它包含两个主要功能：

- **从文件读取员工信息：该功能能够从指定的文件中读取所有员工的信息，并将其传递给员工信息模块进行处理。**
- **向文件写入员工信息：该功能能够将员工信息模块中的员工信息保存到指定的文件中，以便于下次程序运行时能够直接读取。**

2. 员工信息模块：该模块主要负责对员工信息进行管理。它包括以下功能：

- **添加员工：该功能能够向系统中添加新的员工，输入员工的姓名、ID、性别、入职时间、职称、部门等信息。**
- **删除员工：该功能用于删除指定员工的信息，在执行前需要先确认是否真的要删除该员工的信息。**
- **根据员工 ID 查询员工信息：该功能为管理员提供了一种查找特定员工信息的方法，管理员可以输入员工的 ID，以获取该员工的详细信息。**
- **统计各职称员工信息：该功能可以统计不同职称的员工数量，并显示在屏幕上。**
- **统计各部门的员工数量：该功能可以统计不同部门的员工数量，并显示在屏幕上。**
- **根据不同的字段信息排序显示：该功能能够根据管理员选择的字段（如姓名、ID、职称等）对员工信息进行排序，并按照指定顺序显示在屏幕上。**
- **显示所有员工信息：该功能将会在屏幕上显示所有员工的详细信息，以帮助管理员快速了解公司员工情况。**

3. 命令行交互系统模块：该模块主要负责与用户之间的交互。它包含以下功能：

- **显示功能菜单：该功能用于显示员工管理系统的各项功能，让用户可以直观地了解并选择需要的操作。**

- **接收用户输入：该功能可以接收用户输入的命令或参数信息，并传递给相应的功能模块进行处理。**
- **显示结果信息：该功能可以将员工管理系统的操作结果显示在屏幕上，便于管理员进行查看。**

3.2 系统分析及设计

系统涉及对象有两个基本类：管理系统类和员工信息节点类。涉及的功能操作归纳为如下表 1 所示：

表 1 人员类涉及的操作

对象	涉及的对象操作
员工信息节点类	存取基本信息
管理系统链表类	增删改查员工信息
	员工信息统计排序
	读写文件存入链表

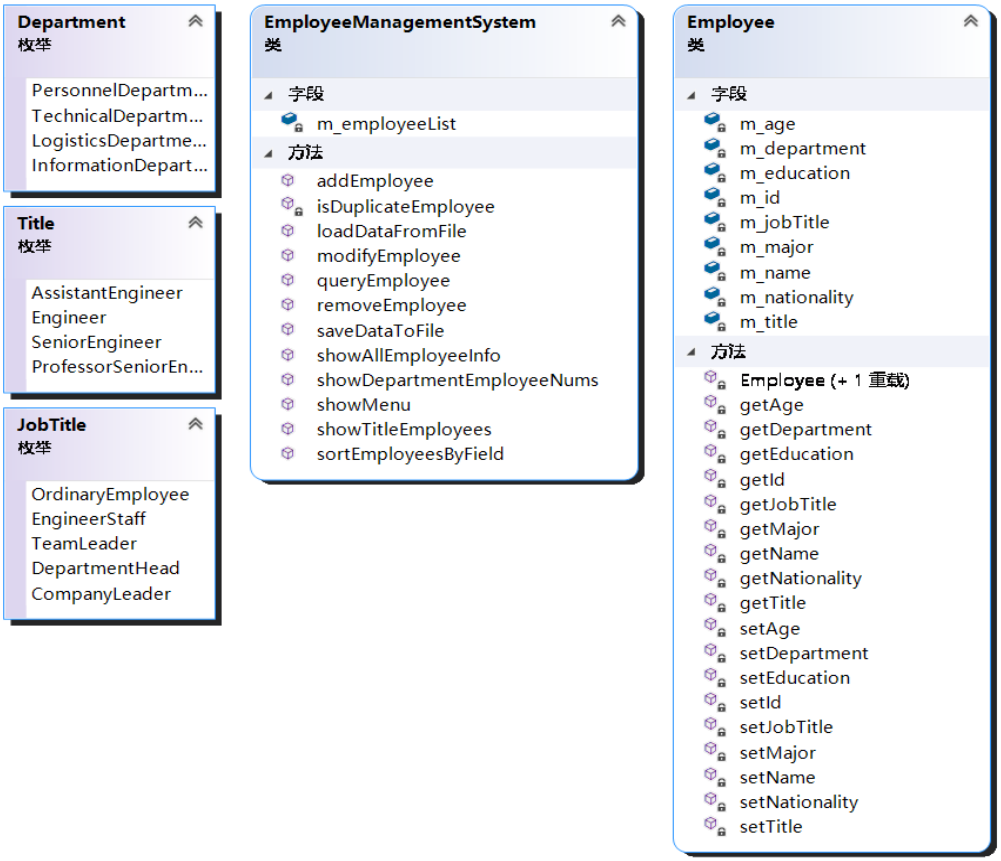


图 2 系统主要类结构图

可以采用面向对象的方式实现职员管理系统。系统的主要的类结构如图 2 所示。

3.3 系统的实现

（1）类的编写

系统工程名为：EMS。包含了 Employee 类（职工节点基类），Employee Management System 类（管理系统链表类）。为了实现对链表类对节点类的访问，我们梳理清楚它们的关系，在节点类的声明中声明管理链表类为友元类。

具体类结构声明如下：

```
// 职称枚举类型
enum Title {
    AssistantEngineer,
    Engineer,
    SeniorEngineer,
    ProfessorSeniorEngineer
};

// 部门枚举类型
enum Department {
    PersonnelDepartment,
    TechnicalDepartment,
    LogisticsDepartment,
    InformationDepartment
};

// 岗位枚举类型
enum JobTitle {
    OrdinaryEmployee,
    EngineerStaff,
    TeamLeader,
    DepartmentHead,
    CompanyLeader
};

// 员工信息类
class Employee {

    friend class EmployeeManagementSystem;

private:
    Employee() = default;
    Employee(int id, string name, int age, string education, string
nationality, string major, Title title, Department department, JobTitle
jobTitle)
```

```
        : m_id(id), m_name(name), m_age(age), m_education(education),
m_nationality(nationality), m_major(major), m_title(title),
m_department(department), m_jobTitle(jobTitle) {}

    // Getter and Setter
    int getId() const { return m_id; }
    void setId(int id) { m_id = id; }

    string getName() const { return m_name; }
    void setName(string name) { m_name = name; }

    int getAge() const { return m_age; }
    void setAge(int age) { m_age = age; }

    string getEducation() const { return m_education; }
    void setEducation(string education) { m_education = education; }

    string getNationality() const { return m_nationality; }
    void setNationality(string nationality) { m_nationality =
nationality; }

    string getMajor() const { return m_major; }
    void setMajor(string major) { m_major = major; }

    Title getTitle() const { return m_title; }
    void setTitle(Title title) { m_title = title; }

    Department getDepartment() const { return m_department; }
    void setDepartment(Department department) { m_department =
department; }

    JobTitle getJobTitle() const { return m_jobTitle; }
    void setJobTitle(JobTitle jobTitle) { m_jobTitle = jobTitle; }

    int m_id;
    string m_name;
    int m_age;
    string m_education;
    string m_nationality;
    string m_major;
    Title m_title;
    Department m_department;
    JobTitle m_jobTitle;
};
```

```
// 员工管理系统类
class EmployeeManagementSystem {
public:
    // 菜单函数
    void showMenu();

    // 文件读写函数
    void saveDataToFile();
    void loadDataFromFile();

    // 操作员工数据函数
    void addEmployee();
    void modifyEmployee();
    void removeEmployee();
    void queryEmployee();
    void showTitleEmployees();
    void showDepartmentEmployeeNums();
    void sortEmployeesByField();
    void showAllEmployeeInfo();


private:
    bool isDuplicateEmployee(int id) const;
    list<Employee> m_employeeList;
};
```

（2）链表的使用

系统实现采用文件的输入输出流对文本数据进行读取与写入，但是由于职工信息是一个数据的集合，于是对数据的存储组织使用了双向链表。

因为职工管理系统在查找、修改、添加、删除的时候都需要处理大量不同员工的数据，所以使用链表十分必要。我们在 Book 类的基础上定义一个对应的元素 m_employeeList 来管理员工数据，具体的结构声明如下：

```
list<Employee> m_employeeList;
```

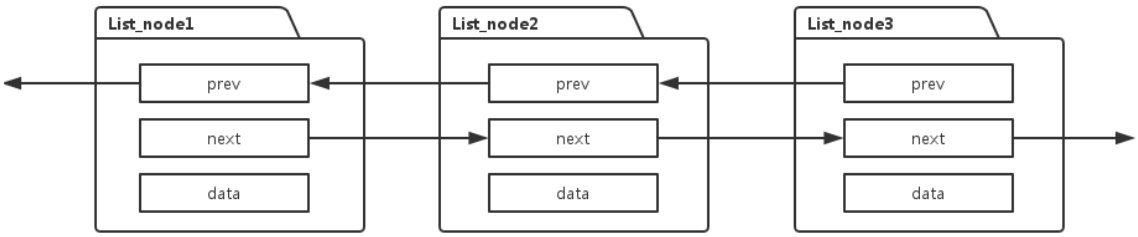


图 3 *职工链表的建立*

职工管理系统的信息的管理就具体表现为职工链表的操作。职工信息的查找、修改、添加和删除与链表的查找、修改、添加、删除对应。

- **查询员工：**

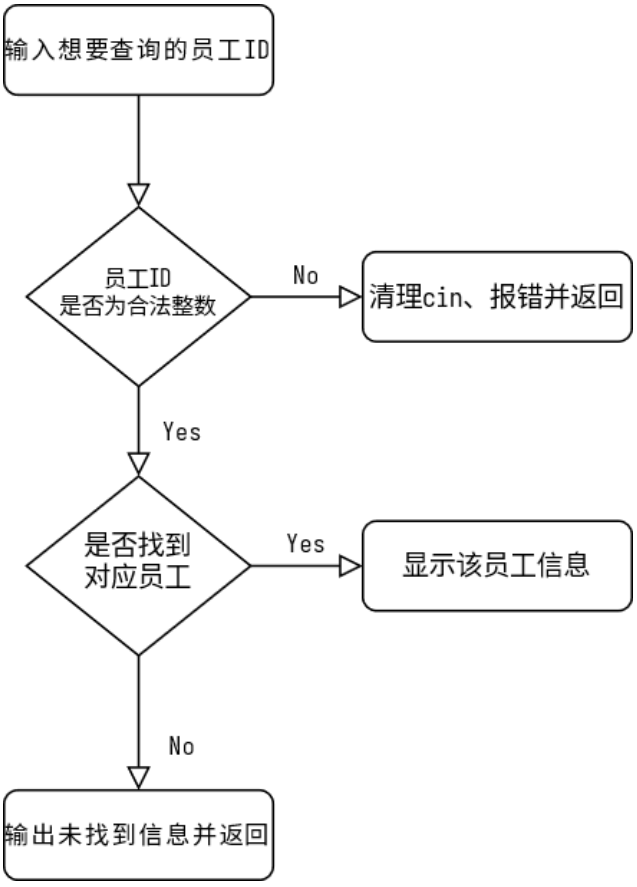
```
    int id;
    cout << "请输入要查询的员工 ID: ";
    cin >> id;

    auto it = find_if(m_employeeList.begin(), m_employeeList.end(),
[id](const Employee& e) { return e.getId() == id; });

    if (it == m_employeeList.end()) {
        cout << "没有找到 ID 为" << id << "的员工" << endl;
        return;
    }
```

程序首先打印提示信息让用户输入要查询的员工 ID，然后读取用户输入的 ID 保存到变量 id 中。如果输入的值产生错误，那么便会返回错误信息并退出函数。

接下来，程序使用 STL 算法库中的 find_if 函数在 m_employeeList 列表中查找一个满足条件的元素：元素的 getId()函数返回值等于 id。这里使用了 C++11 引入的 Lambda 表达式，其中[id]表示捕获 id 变量，即在 Lambda 表达式中可以使用外部变量 id，而(const Employee& e)则表示函数参数 e 是 const Employee 型的常引用类型。如果查找成功，则返回指向该元素的迭代器 it，否则 it 等于 m_employeeList.end()。

最后，程序根据 it 是否等于 m_employeeList.end()来判断是否成功查找到了对应的员工，如果没有找到则输出相应提示信息，否则输出找到的员工信息。

员工查找的流程图如下：



- **添加员工：**

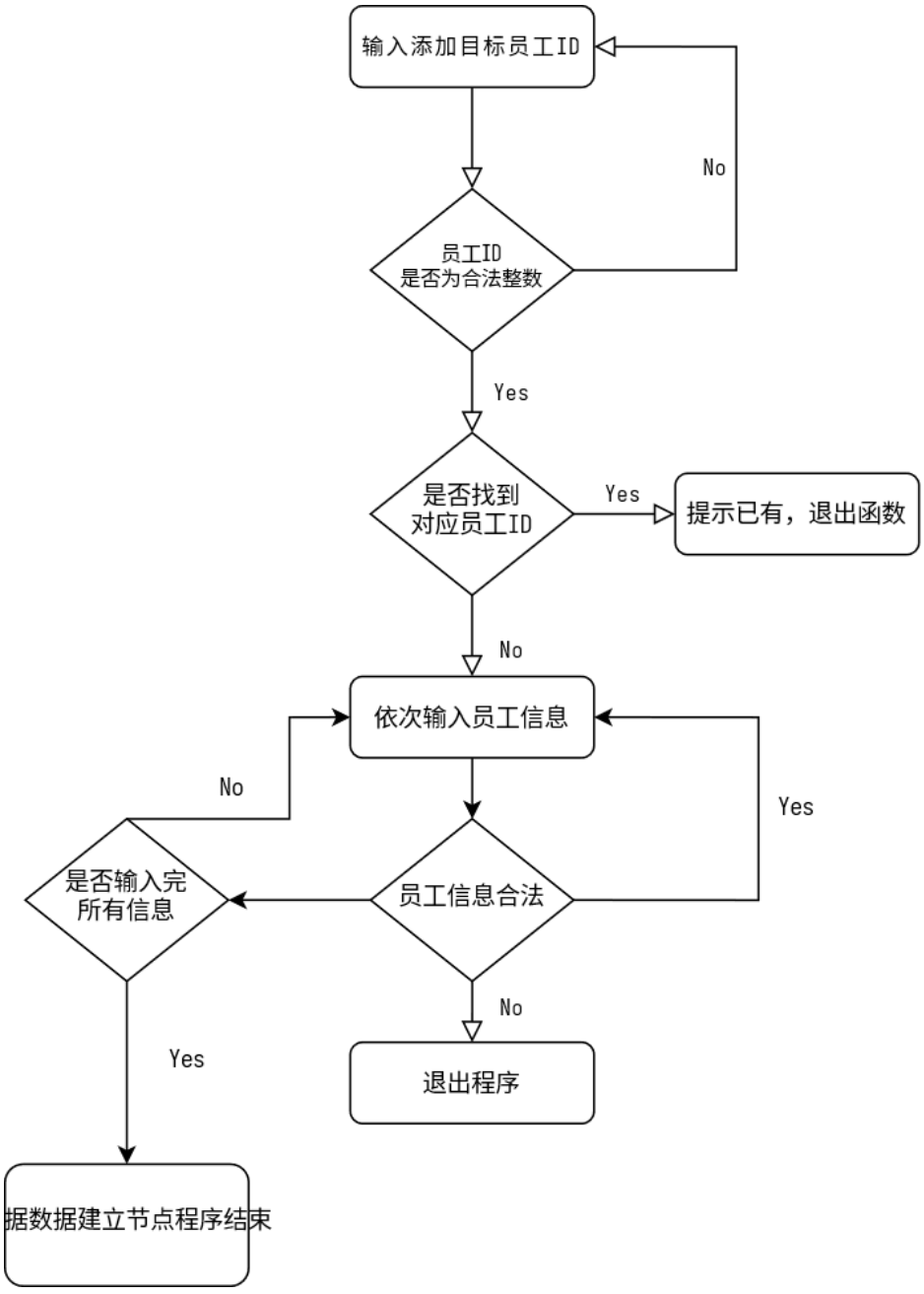
这段程序是用来添加员工数据的。具体来说，程序先提示用户输入员工的 ID，如果输入不合法则会反复提示要求重新输入。

然后，程序检查该 ID 是否已经存在于 m_employeeList 列表中，如果存在则输出提示信息并退出函数。

接着，程序依次询问用户输入员工的姓名、年龄、受教育程度、民族、专业、职称、部门和岗位，并对输入进行合法性检查。其中，职称、部门和岗位都是使用枚举类型表示的，程序通过将用户输入的数字转换为枚举类型值来保存对应的职称、部门和岗位信息。

```
 m_employeeList.emplace_back(Employee(id, name, age, education,
nationality, major, title, department, jobTitle));
```

最后，程序将输入的信息封装成一个 Employee 对象，并添加到 m_employeeList 列表末尾，表示新员工入职成功。若添加成功，则输出提示信息。



- **其他函数：**

运用到了链表的函数还包括修改员工信息、删除员工、查询员工、显示各职称的员工信息，显示各部门的员工数量，根据不同的字段排序，显示所有员工的信息，从文件读取员工信息，向文件写入员工信息等。它们函数中与链表相关的程序实现如下：

写入文件（若无重复 id 则写入）

```
        if (id != 0 && !isDuplicateEmployee(id)) {
            Employee employee(id, name, age, education, nationality, major,
title, dpt, jobTitle);
            m_employeeList.push_back(employee);
        }
```

读取文件（遍历）

```
for (auto& emp : m_employeeList)
```

删除员工信息（找到并抹去节点）

```
auto it = find_if(m_employeeList.begin(), m_employeeList.end(), [id](const
Employee& e) { return e.getId() == id; });
m_employeeList.erase(it);
```

展示所有员工的信息（遍历）

```
for (const auto& employee : m_employeeList)
```

显示员工职称

（定义 map 来存放员工列表）

```
void EmployeeManagementSystem::showTitleEmployees()
{

    // 定义一个 map，用于存放各职称对应的员工列表
    map<Title, vector<Employee>> titleEmployeesMap;
    for (const auto& emp : m_employeeList)
    {
        // 将员工按照职称分组
        titleEmployeesMap[emp.getTitle()].emplace_back(emp);
    }

    // 遍历 map，输出各职称的员工数量和员工信息
    for (const auto& item : titleEmployeesMap)
    {
        Title title = item.first;
        for (const auto& emp : item.second)
        {
        }
    }
}
```

显示各部门员工数量

```
    cout << "人事部: " << count_if(m_employeeList.begin(),
m_employeeList.end(), [](const Employee& e) { return e.getDepartment() ==
PersonnelDepartment; }) << endl;
```

根据不同的字段排序显示员工信息

```
 m_employeeList.sort([](const Employee& e1, const Employee& e2) { return
e1.getId() < e2.getId(); });
```

（3）交互界面以及登录菜单的实现

系统运行开始的界面如图所示：

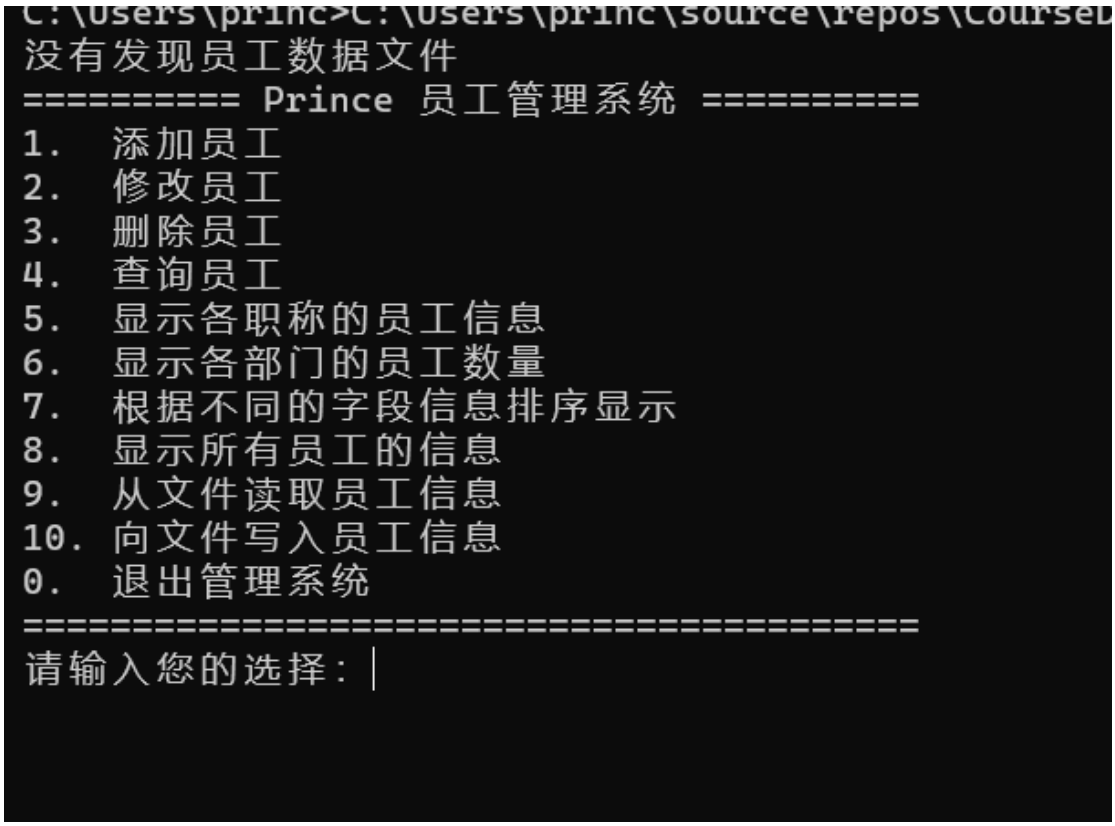
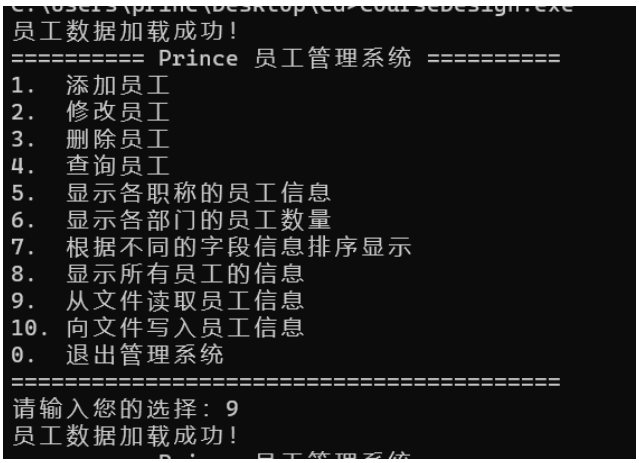


图 4 *开始界面*

主要通过 switch 选择结构和循环结构实现界面的前进和后退。例如，第一个登录界面出现 11 个选择，囊括了所有功能的选择。

四、 实验调试、测试、运行记录及分析

进入界面，选择读取员工信息，读取成功：



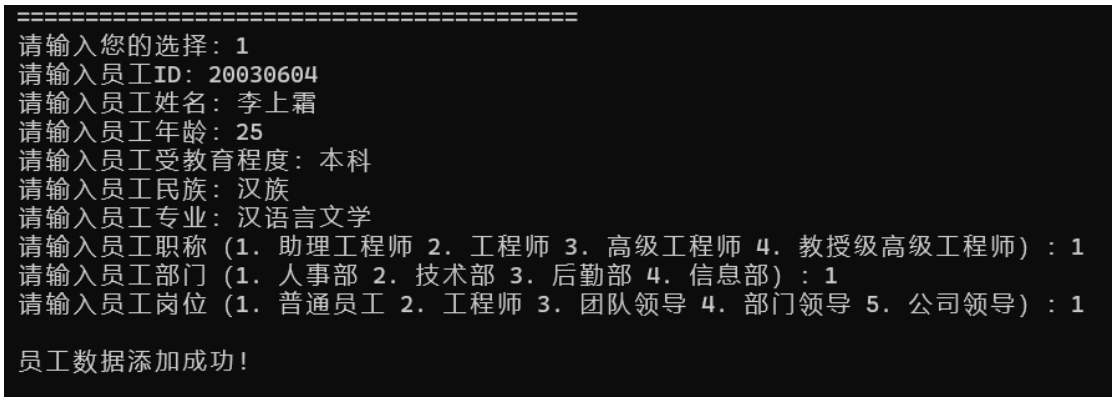
```
员工数据加载成功!
========== Prince 员工管理系统 ==========
1.  添加员工
2.  修改员工
3.  删除员工
4.  查询员工
5.  显示各职称的员工信息
6.  显示各部门的员工数量
7.  根据不同的字段信息排序显示
8.  显示所有员工的信息
9.  从文件读取员工信息
10. 向文件写入员工信息
0.  退出管理系统
========================================
请输入您的选择: 8

所有员工信息如下:
--------------------------------
编号: 20021012
姓名: 洛日远
年龄: 23
受教育程度: 本科
民族: 汉族
专业: 计算机类
职称: 助理工程师
部门: 人事部
岗位: 普通员工
--------------------------------
编号: 20211201
姓名: 高生与
年龄: 20
受教育程度: 本科
```

员工信息修改成功：

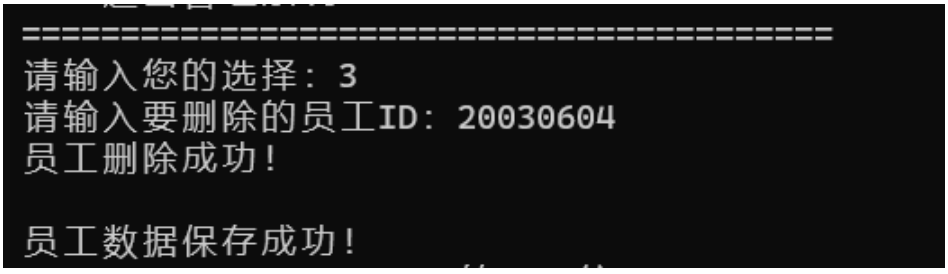
```
--------------------------------
========== Prince 员工管理系统 ==========
1.  添加员工
2.  修改员工
3.  删除员工
4.  查询员工
5.  显示各职称的员工信息
6.  显示各部门的员工数量
7.  根据不同的字段信息排序显示
8.  显示所有员工的信息
9.  从文件读取员工信息
10. 向文件写入员工信息
0.  退出管理系统
========================================
请输入您的选择: 2
请输入要修改的员工ID: 20021012
请输入要修改的字段 (1. 姓名 2. 年龄 3. 受教育程度 4. 民族 5. 专业 6. 职称 7. 部门 8. 岗位)
请输入新的年龄: 29

员工信息修改成功!
```

员工信息查询成功：

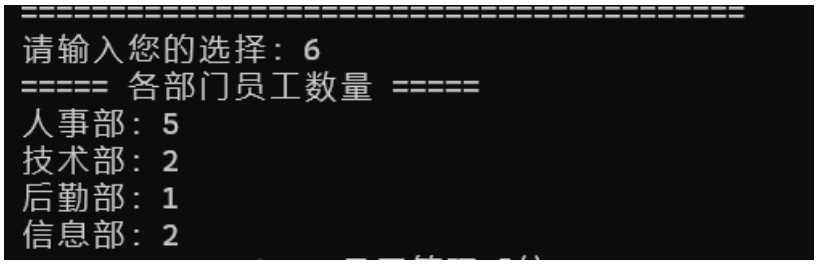
员工信息添加成功：



员工信息删除成功：



分别统计各部门员工数量成功：



分别展示各职称的员工信息成功：

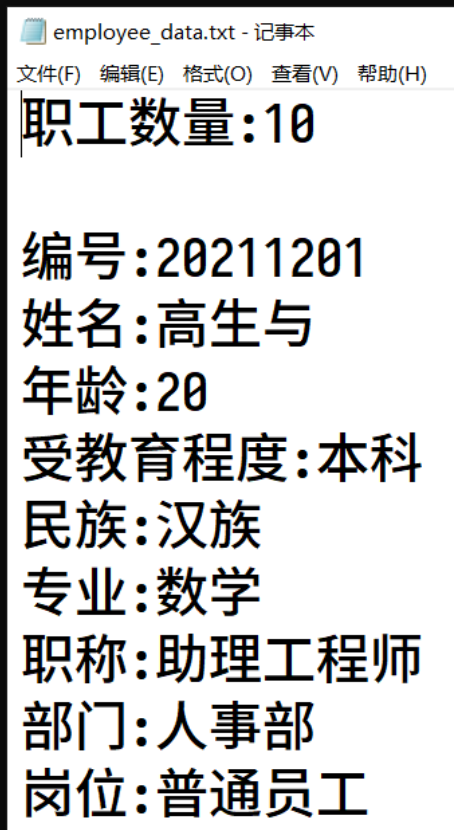
```
==================================
请输入您的选择：5
===== 各职称的员工信息 =====

职称：助理工程师        员工数量：5
助理工程师员工信息：
        编号：20021012  姓名：洛日远    年龄：29        学历：本科      籍贯：汉族      专业：计算机类
        编号：20211201  姓名：高生与    年龄：20        学历：本科      籍贯：汉族      专业：数学
        编号：20111018  姓名：刘瑟瑟    年龄：30        学历：本科      籍贯：汉族      专业：文学与新闻传媒类
        编号：20121015  姓名：张楚      年龄：32        学历：本科      籍贯：汉族      专业：历史学类
        编号：20181201  姓名：李霓裳    年龄：29        学历：本科      籍贯：汉族      专业：化学工程与工艺

职称：工程师    员工数量：1
工程师员工信息：
        编号：20041027  姓名：李清秋    年龄：25        学历：硕士研究生        籍贯：汉族      专业：计算机类

职称：高级工程师        员工数量：3
高级工程师员工信息：
        编号：20061002  姓名：王千岩    年龄：28        学历：博士研究生        籍贯：回族      专业：计算机类
        编号：20140508  姓名：吴冠贤    年龄：40        学历：硕士研究生        籍贯：汉族      专业：金融学类
        编号：20151111  姓名：江近月    年龄：38        学历：本科      籍贯：汉族      专业：市场营销类

职称：教授级高级工程师  员工数量：1
教授级高级工程师员工信息：
        编号：20170123  姓名：吴现世    年龄：45        学历：硕士研究生        籍贯：汉族      专业：自动化
```

示例，按年龄排序，排序成功：10

```
==========================================
请输入您的选择：7
请输入排序字段：1.编号 2.姓名 3.年龄 4.职称 5.部门 6.岗位：3
根据员工年龄排序。
员工信息如下：
------------------------------
编号：20211201
姓名：高生与
年龄：20
受教育程度：本科
民族：汉族
专业：数学
职称：助理工程师
部门：人事部
岗位：普通员工
------------------------------
编号：20041027
姓名：李清秋
年龄：25
受教育程度：硕士研究生
民族：汉族
专业：计算机类
职称：工程师
部门：后勤部
岗位：工程师
------------------------------
编号：20061002
姓名：王千岩
年龄：28
受教育程度：博士研究生
民族：回族
专业：计算机类
职称：高级工程师
部门：信息部
岗位：团队组长
------------------------------
编号：20021012
姓名：洛日远
年龄：29
```

写入文件的功能测试成功：

```
编号: 20170123
姓名: 吴现世
年龄: 45
受教育程度: 硕士研究生
民族: 汉族
专业: 自动化
职称: 教授级高级工程师
部门: 技术部
岗位: 部门负责人
----------------------------
========== Prince 员工管理系统 ==========
1.  添加员工
2.  修改员工
3.  删除员工
4.  查询员工
5.  显示各职称的员工信息
6.  显示各部门的员工数量
7.  根据不同的字段信息排序显示
8.  显示所有员工的信息
9.  从文件读取员工信息
10. 向文件写入员工信息
0.  退出管理系统
========================================
请输入您的选择: 10

员工数据保存成功!
========== Prince 员工管理系统 ==========
1.  添加员工
2.  修改员工
3.  删除员工
4.  查询员工
5.  显示各职称的员工信息
6.  显示各部门的员工数量
```



遇到的问题及解决方法如下：

- **问题 1：**

问题描述：

一开始程序在遇到意外的输入时（比如 ID 输入字符串）会陷入死循环。

解决方法：

```
while (cin.fail()) {
        cin.clear();            // 清除错误状态
        cin.ignore(100, '\n');  // 消耗掉输入流中的错误字符
        cout << "输入有误，请重新输入员工 ID: ";
        cin >> id;
    }
```

使用以上函数进行合法性检验。

实现了对非法输入的纠错：

```
==========================================
请输入您的选择：1
请输入员工ID：hahahahashah
输入有误，请重新输入员工ID：hdfasds
输入有误，请重新输入员工ID：20210315
请输入员工姓名：邱山
请输入员工年龄：34
请输入员工受教育程度：本科
请输入员工民族：汉族
请输入员工专业：电子信息技术
请输入员工职称（1. 助理工程师 2. 工程师 3. 高级工程师 4. 教授级高级工程师）：2
请输入员工部门（1. 人事部 2. 技术部 3. 后勤部 4. 信息部）：2
请输入员工岗位（1. 普通员工 2. 工程师 3. 团队领导 4. 部门领导 5. 公司领导）：2

员工数据添加成功！
```

图　调试测试问题 1

- **问题 2：**

问题描述：

当读取文件时，涉及空缺的职员数据时无法读入，会爆出内存错误。

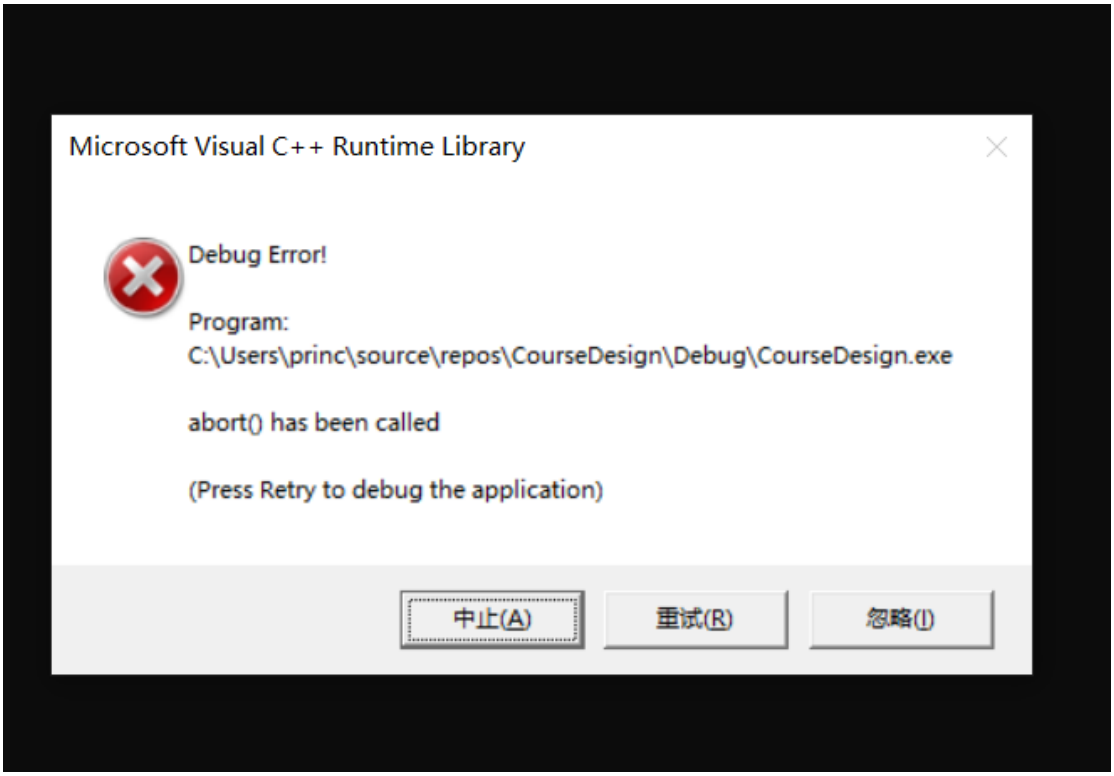
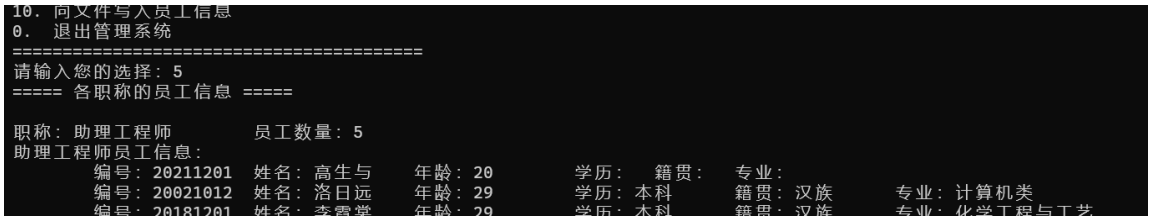


图 6　调试测试问题 2

解决办法：

重写了读入函数，使得可以正常读入空缺的数据。

```
    getline(inFile, line); // 读取空行
    getline(inFile, line); // 读取 id
    int id = stoi(line.substr(line.find(':') + 1));
```



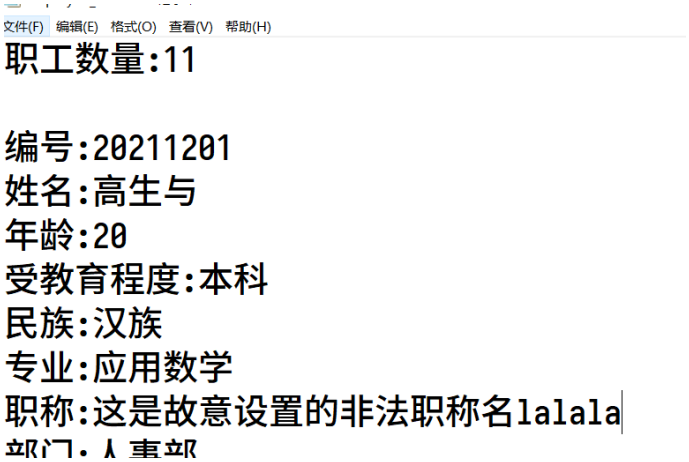
- **问题 3：**

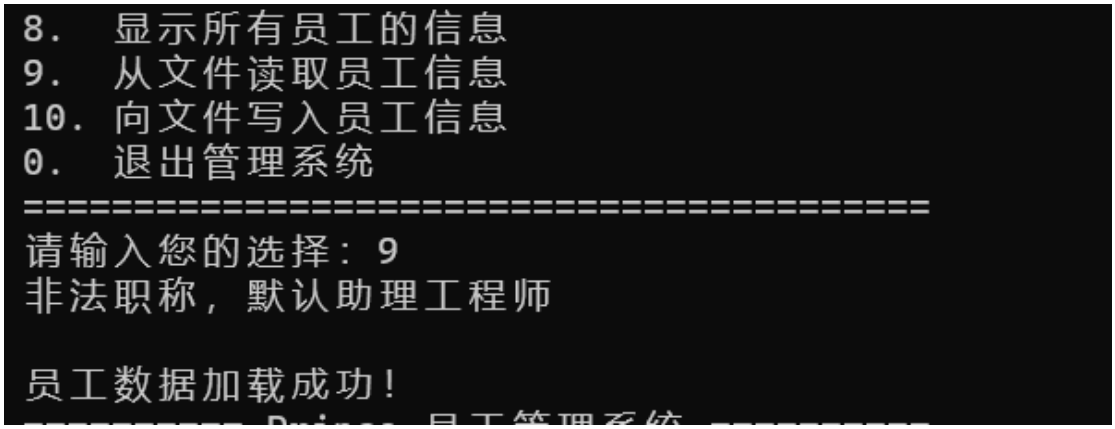
问题描述：

会不加判断地读入非法职称、非法部门和非法岗位。

解决方法：

设置了面对非法职称、部门、岗位的例外判断情况。





五、 实验总结

在本次实验中,我学习了如何设计一个简单的员工管理系统,该系统主要包含三个模块:读写文件模块、员工信息模块和命令行交互系统模块。通过对这些模块的了解和实现,我掌握了以下几个方面的知识和经验。

优点

- 功能完整:该员工管理系统的功能包括添加、删除、修改、查询员工信息等,能够满足基本的员工管理需求。
- 界面友好:通过命令行界面进行交互,用户可以直观地了解到系统提供的各项功能,并且可以通过输入相应的命令和参数来操作系统,使用起来比较便捷。
- 维护简单:该系统不需要复杂的技术支持,只需要掌握基本的编程知识就可以轻松维护。

不足

- 数据存储方式简单:该系统采用文本文件来存储员工信息,无法处理大量数据,也不太安全。
- 功能单一:虽然该系统包含基本的员工管理功能,但是相比较于实际企业的员工管理需求,还远远不够完善。

收获与体会

通过本次实验,我体会到了开发一个程序的基本流程,包括需求分析、系统设计、编码实现、测试和维护等环节。同时,我也了解到了C++编程语言的一些基本语法和操作方式,掌握了使用文件读写模块、函数定义、异常处理和命令行交互等编程技巧。虽然该系统还有许多不足之处,但是这次实验为我日后进一步学习和探索员工管理系统提供了基础和启示。

六、 附录:源代码

```
#include <iostream>
#include <fstream>
#include <string>
#include <list>
#include <map>
#include <vector>
```

```
#include <algorithm>
#include<io.h>//使用_setmode(_fileno(stdout), _O_U16TEXT)必须加的头文件
#include<fcntl.h>//使用_setmode(_fileno(stdout), _O_U16TEXT)必须加的头文件
using namespace std;

// 职称枚举类型
enum Title {
    AssistantEngineer,
    Engineer,
    SeniorEngineer,
    ProfessorSeniorEngineer
};

// 部门枚举类型
enum Department {
    PersonnelDepartment,
    TechnicalDepartment,
    LogisticsDepartment,
    InformationDepartment
};

// 岗位枚举类型
enum JobTitle {
    OrdinaryEmployee,
    EngineerStaff,
    TeamLeader,
    DepartmentHead,
    CompanyLeader
};

// 获取职称对应的中文名称
string getTitleChinese(Title title) {
    switch (title) {
    case AssistantEngineer:
        return "助理工程师";
    case Engineer:
        return "工程师";
    case SeniorEngineer:
        return "高级工程师";
    case ProfessorSeniorEngineer:
        return "教授级高级工程师";
    default:
        return "";
    }
```

```
}


// 获取部门对应的中文名称
string getDepartmentChinese(Department department) {
    switch (department) {
    case PersonnelDepartment:
        return "人事部";
    case TechnicalDepartment:
        return "技术部";
    case LogisticsDepartment:
        return "后勤部";
    case InformationDepartment:
        return "信息部";
    default:
        return "";
    }
}

// 获取岗位对应的中文名称
string getJobTitleChinese(JobTitle jobTitle) {
    switch (jobTitle) {
    case OrdinaryEmployee:
        return "普通员工";
    case EngineerStaff:
        return "工程师";
    case TeamLeader:
        return "团队组长";
    case DepartmentHead:
        return "部门负责人";
    case CompanyLeader:
        return "公司领导";
    default:
        return "";
    }
}


Title getTitleEnum(const string& titleStr)
{
    if (titleStr == "助理工程师"||titleStr=="1") {
        return Title::AssistantEngineer;
    }
    else if (titleStr == "工程师"||titleStr == "2") {
        return Title::Engineer;
```

```
    }
    else if (titleStr == "高级工程师" || titleStr == "3") {
        return Title::SeniorEngineer;
    }
    else if (titleStr == "教授级高级工程师" || titleStr == "4") {
        return Title::ProfessorSeniorEngineer;
    }
    else {
        cout<<"非法职称，默认助理工程师\n"<<endl;
        return Title::AssistantEngineer;
    }
}


Department getDepartmentEnum(const string& dptStr)
{
    if (dptStr == "人事部"||dptStr == "1") {
        return Department::PersonnelDepartment;
    }
    else if (dptStr == "技术部" || dptStr == "2") {
        return Department::TechnicalDepartment;
    }
    else if (dptStr == "后勤部" || dptStr == "3") {
        return Department::LogisticsDepartment;
    }
    else if (dptStr == "信息部" || dptStr == "4") {
        return Department::InformationDepartment;
    }
    else {
        cout << "非法部门，默认人事部\n" << endl;
        return Department::PersonnelDepartment;
    }
}

JobTitle getJobTitleEnum(const string& jobStr)
{
    if (jobStr == "普通员工"|| jobStr == "1") {
        return JobTitle::OrdinaryEmployee;
    }
    else if (jobStr == "工程师" || jobStr == "2") {
        return JobTitle::EngineerStaff;
    }
    else if (jobStr == "团队组长" || jobStr == "3") {
        return JobTitle::TeamLeader;
    }
```

```
    else if (jobStr == "部门负责人" || jobStr == "4") {
        return JobTitle::DepartmentHead;
    }
    else if (jobStr == "公司领导" || jobStr == "5") {
        return JobTitle::CompanyLeader;
    }
    else {
        cout << "非法岗位，默认普通员工\n" << endl;
        return JobTitle::OrdinaryEmployee;
    }
}



// 员工信息类
class Employee {

    friend class EmployeeManagementSystem;

private:
    Employee() = default;
    Employee(int id, string name, int age, string education, string
nationality, string major, Title title, Department department, JobTitle
jobTitle)
        : m_id(id), m_name(name), m_age(age), m_education(education),
m_nationality(nationality), m_major(major), m_title(title),
m_department(department), m_jobTitle(jobTitle) {}

    // Getter and Setter
    int getId() const { return m_id; }
    void setId(int id) { m_id = id; }

    string getName() const { return m_name; }
    void setName(string name) { m_name = name; }

    int getAge() const { return m_age; }
    void setAge(int age) { m_age = age; }

    string getEducation() const { return m_education; }
    void setEducation(string education) { m_education = education; }

    string getNationality() const { return m_nationality; }
    void setNationality(string nationality) { m_nationality =
nationality; }
```

```
    string getMajor() const { return m_major; }
    void setMajor(string major) { m_major = major; }

    Title getTitle() const { return m_title; }
    void setTitle(Title title) { m_title = title; }

    Department getDepartment() const { return m_department; }
    void setDepartment(Department department) { m_department =
department; }

    JobTitle getJobTitle() const { return m_jobTitle; }
    void setJobTitle(JobTitle jobTitle) { m_jobTitle = jobTitle; }

    int m_id;
    string m_name;
    int m_age;
    string m_education;
    string m_nationality;
    string m_major;
    Title m_title;
    Department m_department;
    JobTitle m_jobTitle;
};

// 员工管理系统类
class EmployeeManagementSystem {
public:
    // 菜单函数
    void showMenu();

    // 文件读写函数
    void saveDataToFile();
    void loadDataFromFile();

    // 操作员工数据函数
    void addEmployee();
    void modifyEmployee();
    void removeEmployee();
    void queryEmployee();
    void showTitleEmployees();
    void showDepartmentEmployeeNums();
    void sortEmployeesByField();
    void showAllEmployeeInfo();
```

```
private:
    bool isDuplicateEmployee(int id) const;
    list<Employee> m_employeeList;
};

/**
 * @brief 显示菜单
 */

void EmployeeManagementSystem::showMenu()
{
    std::cout << "========== Prince 员工管理系统 ==========" << std::endl
        << "1.  添加员工" << std::endl
        << "2.  修改员工" << std::endl
        << "3.  删除员工" << std::endl
        << "4.  查询员工" << std::endl
        << "5.  显示各职称的员工信息" << std::endl
        << "6.  显示各部门的员工数量" << std::endl
        << "7.  根据不同的字段信息排序显示" << std::endl
        << "8.  显示所有员工的信息" << std::endl
        << "9.  从文件读取员工信息" << std::endl
        << "10. 向文件写入员工信息" << std::endl
        << "0.  退出管理系统" << std::endl
        << "=========================================" << std::endl
        << "请输入您的选择: ";
}



/**
 * @brief 根据 ID 看链表中是否有重复 ID
 */
bool EmployeeManagementSystem::isDuplicateEmployee(int id) const
{
    for (auto& emp : m_employeeList) {
        if (emp.getId() == id) {
            return true;
        }
    }
    return false;
}

/**
```

```
 * @brief 从文件中读取数据
 */
void EmployeeManagementSystem::loadDataFromFile()
{
    ifstream inFile("employee_data.txt");
    if (!inFile) {
        cout << "没有发现员工数据文件\n";
        ofstream outFile("employee_data.txt");
        if (!outFile) {
            cout << "无法创建全新的员工数据文件，数据保存失败！" << endl;
            return;
        }
        return;
    }

    string line;
    getline(inFile, line);
    int empNum=stoi(line.substr(line.find(':') + 1));//读职工数量

    for (int i = 0; i < empNum; ++i) {
        getline(inFile, line); // 读取空行
        getline(inFile, line); // 读取 id
        int id = stoi(line.substr(line.find(':') + 1));

        getline(inFile, line); // 读取 name
        string name = line.substr(line.find(':') + 1);

        getline(inFile, line); // 读取 age
        int age = stoi(line.substr(line.find(':') + 1));

        getline(inFile, line); // 读取 education
        string education = line.substr(line.find(':') + 1);

        getline(inFile, line); // 读取 nationality
        string nationality = line.substr(line.find(':') + 1);

        getline(inFile, line); // 读取 major
        string major = line.substr(line.find(':') + 1);

        getline(inFile, line); // 读取 title
        string titleStr = line.substr(line.find(':') + 1);
        Title title = getTitleEnum(titleStr);

        getline(inFile, line); // 读取 department
```

```
        string dptStr = line.substr(line.find(':') + 1);
        Department dpt = getDepartmentEnum(dptStr);

        getline(inFile, line); // 读取 jobTitle
        string jobStr = line.substr(line.find(':') + 1);
        JobTitle jobTitle = getJobTitleEnum(jobStr);

        if (id != 0 && !isDuplicateEmployee(id)) {
            Employee employee(id, name, age, education, nationality, major,
title, dpt, jobTitle);
            m_employeeList.push_back(employee);
        }
    }

    cout << "员工数据加载成功！" << endl;
}



/**
 * @brief 往文件中写数据
 */
void EmployeeManagementSystem::saveDataToFile()
{
    ofstream outFile("employee_data.txt", std::ofstream::trunc);
    if (!outFile) {
        cout << "无法创建员工数据文件，数据保存失败！" << endl;
        return;
    }

    // 写入员工数量
    int empNum = m_employeeList.size();
    outFile << "职工数量:" << empNum << endl;

    // 依次写入每个员工的信息
    for (auto& emp : m_employeeList) {
        outFile << endl;   // 空行
        outFile << "编号:" << emp.getId() << endl;
        outFile << "姓名:" << emp.getName() << endl;
        outFile << "年龄:" << emp.getAge() << endl;
        outFile << "受教育程度:" << emp.getEducation() << endl;
        outFile << "民族:" << emp.getNationality() << endl;
        outFile << "专业:" << emp.getMajor() << endl;
        outFile << "职称:" << getTitleChinese(emp.getTitle()) << endl;
```

```
        outFile << "部门:" << getDepartmentChinese(emp.getDepartment()) <<
endl;
        outFile << "岗位:" << getJobTitleChinese(emp.getJobTitle()) <<
endl;
    }

    cout << "\n 员工数据保存成功！" << endl;
}




/**
 * @brief 添加员工信息
 */
void EmployeeManagementSystem::addEmployee()
{
    int id, age;
    string name, education, nationality, major;
    Title title;
    Department department;
    JobTitle jobTitle;

    cout << "请输入员工 ID：";
    cin >> id;
    while (cin.fail()) {
        cin.clear();            // 清除错误状态
        cin.ignore(100, '\n');  // 消耗掉输入流中的错误字符
        cout << "输入有误，请重新输入员工 ID：";
        cin >> id;
    }

    if (isDuplicateEmployee(id)) {
        cout << "员工 ID 已经存在，请重新输入。" << endl;
        return;
    }


    cout << "请输入员工姓名：";
    cin >> name;

    cout << "请输入员工年龄：";
    cin >> age;

    cout << "请输入员工受教育程度：";
```

```
    cin >> education;

    cout << "请输入员工民族：";
    cin >> nationality;

    cout << "请输入员工专业：";
    cin >> major;

    cout << "请输入员工职称（1. 助理工程师 2. 工程师 3. 高级工程师 4. 教授级高
级工程师）：";
    int titleNum;
    cin >> titleNum;

    while (titleNum < 1 || titleNum > 4||cin.fail()) {
        cin.clear();            // 清除错误状态
        cin.ignore(100, '\n');  // 消耗掉输入流中的错误字符
        cout << "您需要输入一个 1-4 的数字，请重新输入员工职称编号：";
        cin >> titleNum;
    }

    title = static_cast<Title>(titleNum - 1);

    cout << "请输入员工部门（1. 人事部 2. 技术部 3. 后勤部 4. 信息部）：";
    int departmentNum;
    cin >> departmentNum;

    while (departmentNum < 1 || departmentNum > 4||cin.fail()) {
        cin.clear();            // 清除错误状态
        cin.ignore(100, '\n');  // 消耗掉输入流中的错误字符
        cout << "您需要输入一个 1-4 的数字。\n 请重新输入部门职称编号：";
        cin >> departmentNum;
    }


    department = static_cast<Department>(departmentNum - 1);

    cout << "请输入员工岗位（1. 普通员工 2. 工程师 3. 团队领导 4. 部门领导 5.
公司领导）：";
    int jobTitleNum;
    cin >> jobTitleNum;

    if (jobTitleNum < 1 || jobTitleNum > 5||cin.fail()) {
        cin.clear();            // 清除错误状态
        cin.ignore(100, '\n');  // 消耗掉输入流中的错误字符
        cout << "您需要输入一个 1-5 的数字。\n 请重新输入员工岗位编号：" << endl;
```

```
        return;
    }

    jobTitle = static_cast<JobTitle>(jobTitleNum - 1);

    m_employeeList.emplace_back(Employee(id, name, age, education,
nationality, major, title, department, jobTitle));

    cout << "\n 员工数据添加成功! \n" << endl;
}

/**
 * @brief 修改员工信息
 */
void EmployeeManagementSystem::modifyEmployee()
{
    int id;
    cout << "请输入要修改的员工 ID: ";
    cin >> id;
    while (cin.fail()) {
        cin.clear();            // 清除错误状态
        cin.ignore(100, '\n');  // 消耗掉输入流中的错误字符
        cout << "输入有误，请重新输入员工 ID: ";
        cin >> id;
    }

    auto it = find_if(m_employeeList.begin(), m_employeeList.end(),
[id](const Employee& e) { return e.getId() == id; });

    if (it == m_employeeList.end()) {
        cout << "没有找到 ID 为" << id << "的员工" << endl;
        return;
    }

    cout << "请输入要修改的字段（1. 姓名 2. 年龄 3. 受教育程度 4. 民族 5. 专业
6. 职称 7. 部门 8. 岗位）: ";
    int fieldNum;
    cin >> fieldNum;

    switch (fieldNum) {
    case 1: {
        string name;
        cout << "请输入新的姓名: ";
        cin >> name;
```

```
        it->setName(name);
        break;
    }
    case 2: {
        int age;
        cout << "请输入新的年龄：";
        cin >> age;
        it->setAge(age);
        break;
    }
    case 3: {
        string education;
        cout << "请输入新的受教育程度：";
        cin >> education;
        it->setEducation(education);
        break;
    }
    case 4: {
        string nationality;
        cout << "请输入新的民族：";
        cin >> nationality;
        it->setNationality(nationality);
        break;
    }
    case 5: {
        string major;
        cout << "请输入新的专业：";
        cin >> major;
        it->setMajor(major);
        break;
    }
    case 6: {
        cout << "请输入新的职称（1. 助理工程师 2. 工程师 3. 高级工程师 4. 教授级高级工程师）：";
        int titleNum;
        cin >> titleNum;

        while (titleNum < 1 || titleNum > 4 || cin.fail()) {
            cin.clear();            // 清除错误状态
            cin.ignore(100, '\n');  // 消耗掉输入流中的错误字符
            cout << "您需要输入一个 1-4 的数字，请重新输入员工职称编号：";
            cin >> titleNum;
        }
```

```
        Title title = static_cast<Title>(titleNum - 1);
        it->setTitle(title);
        break;
    }
    case 7: {
        cout << "请输入员工部门（1. 人事部 2. 技术部 3. 后勤部 4. 信息部）：
";
        int departmentNum;
        cin >> departmentNum;

        while (departmentNum < 1 || departmentNum > 4 || cin.fail()) {
            cin.clear();            // 清除错误状态
            cin.ignore(100, '\n');  // 消耗掉输入流中的错误字符
            cout << "您需要输入一个 1-4 的数字。\n 请重新输入部门职称编号：";
            cin >> departmentNum;
        }

        Department department = static_cast<Department>(departmentNum - 1);
        it->setDepartment(department);

        break;
    }
    case 8: {
        cout << "请输入新的岗位（1. 普通员工 2. 工程师 3. 团队领导 4. 部门领导
5. 公司领导）：";
        int jobTitleNum;
        cin >> jobTitleNum;

        if (jobTitleNum < 1 || jobTitleNum > 5 || cin.fail()) {
            cin.clear();            // 清除错误状态
            cin.ignore(100, '\n');  // 消耗掉输入流中的错误字符
            cout << "您需要输入一个 1-5 的数字。\n 请重新输入员工岗位编号：" <<
endl;
            return;
        }

        JobTitle jobTitle = static_cast<JobTitle>(jobTitleNum - 1);
        it->setJobTitle(jobTitle);
        break;
    }
    default:
        cout << "字段编号无效，请重新输入。" << endl;
        cin.clear();            // 清除错误状态
        cin.ignore(100, '\n');  // 消耗掉输入流中的错误字符
```

```
        return;
    }

    cout << "\n 员工信息修改成功！\n" << endl;
}

/**
 * @brief 删除员工信息
 */
void EmployeeManagementSystem::removeEmployee()
{
    int id;
    cout << "请输入要删除的员工 ID：";
    cin >> id;

    auto it = find_if(m_employeeList.begin(), m_employeeList.end(),
[id](const Employee& e) { return e.getId() == id; });

    if (it == m_employeeList.end()) {
        cout << "没有找到 ID 为" << id << "的员工" << endl;
        return;
    }

    m_employeeList.erase(it);

    cout << "员工删除成功！" << endl;
}


void EmployeeManagementSystem::showAllEmployeeInfo()
{
    cout << "\n 所有员工信息如下：" << endl;
    cout << "------------------------------" << endl;
    for (const auto& employee : m_employeeList) {

        cout << "编号：" << employee.getId() << endl;
        cout << "姓名：" << employee.getName() << endl;
        cout << "年龄：" << employee.getAge() << endl;
        cout << "受教育程度：" << employee.getEducation() << endl;
        cout << "民族：" << employee.getNationality() << endl;
        cout << "专业：" << employee.getMajor() << endl;
        cout << "职称：" << getTitleChinese(employee.getTitle()) << endl;
        cout << "部门：" << getDepartmentChinese(employee.getDepartment())
<< endl;
```

```
        cout << "岗位：" << getJobTitleChinese(employee.getJobTitle()) <<
endl;

        cout << "------------------------------" << endl;
    }
}


/**
 * @brief 查询员工信息
 */
void EmployeeManagementSystem::queryEmployee()
{
    int id;
    cout << "请输入要查询的员工 ID：";
    cin >> id;

    auto it = find_if(m_employeeList.begin(), m_employeeList.end(),
[id](const Employee& e) { return e.getId() == id; });

    if (it == m_employeeList.end()) {
        cout << "没有找到 ID 为" << id << "的员工" << endl;
        return;
    }

    cout << "查询到的员工信息如下：" << endl;
    cout << "------------------------------" << endl;
    cout << "编号：" << it->getId() << endl;
    cout << "姓名：" << it->getName() << endl;
    cout << "年龄：" << it->getAge() << endl;
    cout << "受教育程度：" << it->getEducation() << endl;
    cout << "民族：" << it->getNationality() << endl;
    cout << "专业：" << it->getMajor() << endl;
    cout << "职称：" << getTitleChinese(it->getTitle()) << endl;
    cout << "部门：" << getDepartmentChinese(it->getDepartment() )<< endl;
    cout << "岗位：" << getJobTitleChinese(it->getJobTitle()) << endl;
}

/**
 * @brief 显示各职称的员工信息
 */
```

```
void EmployeeManagementSystem::showTitleEmployees()
{
    cout << "===== 各职称的员工信息 =====" << endl;

    // 定义一个 map，用于存放各职称对应的员工列表
    map<Title, vector<Employee>> titleEmployeesMap;
    for (const auto& emp : m_employeeList)
    {
        // 将员工按照职称分组
        titleEmployeesMap[emp.getTitle()].emplace_back(emp);
    }

    // 遍历 map，输出各职称的员工数量和员工信息
    for (const auto& item : titleEmployeesMap)
    {
        Title title = item.first;
        cout << "\n 职称：" << getTitleChinese(title) << "\t 员工数量：" <<
item.second.size() << endl;
        cout << getTitleChinese(title)<<"员工信息：" << endl;
        for (const auto& emp : item.second)
        {
            cout << "\t 编号：" << emp.getId() << "\t 姓名：" << emp.getName()
<< "\t 年龄：" << emp.getAge() << "\t 学历："
                << emp.getEducation() << "\t 籍贯：" << emp.getNationality()
<< "\t 专业：" << emp.getMajor() << endl;
        }
    }
    cout << endl;
}



/**
 * @brief 显示各部门员工数量
 */
void EmployeeManagementSystem::showDepartmentEmployeeNums()
{
    cout << "===== 各部门员工数量 =====" << endl;
    cout << "人事部：" << count_if(m_employeeList.begin(),
m_employeeList.end(), [](const Employee& e) { return e.getDepartment() ==
PersonnelDepartment; }) << endl;
    cout << "技术部：" << count_if(m_employeeList.begin(),
m_employeeList.end(), [](const Employee& e) { return e.getDepartment() ==
TechnicalDepartment; }) << endl;
```

```
    cout << "后勤部: " << count_if(m_employeeList.begin(),
m_employeeList.end(), [](const Employee& e) { return e.getDepartment() ==
LogisticsDepartment; }) << endl;
    cout << "信息部: " << count_if(m_employeeList.begin(),
m_employeeList.end(), [](const Employee& e) { return e.getDepartment() ==
InformationDepartment; }) << endl;
}
/**
 * @brief 根据不同的字段排序显示员工信息
 */
void EmployeeManagementSystem::sortEmployeesByField()
{
    cout << "请输入排序字段: 1.编号 2.姓名 3.年龄 4.职称 5.部门 6.岗位: ";
    int fieldNum;
    cin >> fieldNum;

    if (fieldNum == 1) {
        cout << "根据员工编号排序。" << endl;
        m_employeeList.sort([](const Employee& e1, const Employee& e2)
{ return e1.getId() < e2.getId(); });
    }
    else if (fieldNum == 2) {
        cout << "根据员工姓名排序。" << endl;
        m_employeeList.sort([](const Employee& e1, const Employee& e2)
{ return e1.getName() < e2.getName(); });
    }
    else if (fieldNum == 3) {
        cout << "根据员工年龄排序。" << endl;
        m_employeeList.sort([](const Employee& e1, const Employee& e2)
{ return e1.getAge() < e2.getAge(); });
    }
    else if (fieldNum == 4) {
        cout << "根据员工职称排序。" << endl;
        m_employeeList.sort([](const Employee& e1, const Employee& e2)
{ return e1.getTitle() < e2.getTitle(); });
    }
    else if (fieldNum == 5) {
        cout << "根据员工部门排序。" << endl;
        m_employeeList.sort([](const Employee& e1, const Employee& e2)
{ return e1.getDepartment() < e2.getDepartment(); });
    }
    else if (fieldNum == 6) {
        cout << "根据员工岗位排序。" << endl;
```

```
        m_employeeList.sort([](const Employee& e1, const Employee& e2)
{ return e1.getJobTitle() < e2.getJobTitle(); });
    }
    else {
        cout << "字段编号无效，请重新输入。" << endl;
        return;
    }

    cout << "员工信息如下：" << endl;
    cout << "------------------------------" << endl;
    for (const auto& employee : m_employeeList) {
        cout << "编号：" << employee.getId() << endl;
        cout << "姓名：" << employee.getName() << endl;
        cout << "年龄：" << employee.getAge() << endl;
        cout << "受教育程度：" << employee.getEducation() << endl;
        cout << "民族：" << employee.getNationality() << endl;
        cout << "专业：" << employee.getMajor() << endl;
        cout << "职称：" << getTitleChinese(employee.getTitle()) << endl;
        cout << "部门：" << getDepartmentChinese(employee.getDepartment())
<< endl;
        cout << "岗位：" << getJobTitleChinese(employee.getJobTitle())<<
endl;
        cout << "------------------------------" << endl;
    }
}

int main() {
    //_setmode(_fileno(stdout), _O_U16TEXT);//让控制台启用 Unicode 16
    //wcout << L"❶ 👥添加员工" << endl;

    EmployeeManagementSystem ems;

    // 加载数据
    ems.loadDataFromFile();

    // 显示菜单
    while (true) {
        ems.showMenu();
        int choice;
        cin >> choice;
        switch (choice) {
        case 0:
            cout << "\n 谢谢您使用 Prince 员工信息管理系统。欢迎下次再来！" <<
endl;
```

```
            ems.saveDataToFile();
            return 0;
        case 1:
            ems.addEmployee();
            ems.saveDataToFile();
            break;
        case 2:
            ems.modifyEmployee();
            ems.saveDataToFile();
            break;
        case 3:
            ems.removeEmployee();
            ems.saveDataToFile();
            break;
        case 4:
            ems.queryEmployee();
            break;
        case 5:
            ems.showTitleEmployees();
            break;
        case 6:
            ems.showDepartmentEmployeeNums();
            break;
        case 7:
            ems.sortEmployeesByField();
            break;
        case 8:
            ems.showAllEmployeeInfo();
            break;
        case 9:
            ems.loadDataFromFile();
            break;
        case 10:
            ems.saveDataToFile();
            break;
        default:
            cout << "输入有误，请重新输入！" << endl;
            break;
        }
    }

    return 0;

}
```