**项目说明文档**

**数据结构课程设计**

**——考试报名系统**

作 者 姓 名： 张梓瀚

学 号： 2051943

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

目 录

[1 分析 - 4 -](#_Toc121209061)

[1.1 背景分析 - 4 -](#_Toc121209062)

[1.2 功能分析 - 4 -](#_Toc121209063)

[2 设计 - 4 -](#_Toc121209064)

[2.1 数据结构设计 - 4 -](#_Toc121209065)

[2.2 类结构设计 - 4 -](#_Toc121209066)

[2.3 类成员与操作设计 - 5 -](#_Toc121209067)

[2.3.1 Student类 - 5 -](#_Toc121209068)

[2.3.2 ListNode类 - 5 -](#_Toc121209069)

[2.3.3 List类 - 6 -](#_Toc121209070)

[2.3.4 ExamRegisterSystem类 - 7 -](#_Toc121209071)

[2.4 系统设计 - 7 -](#_Toc121209072)

[3 实现 - 8 -](#_Toc121209073)

[3.1 插入功能的实现 - 8 -](#_Toc121209074)

[3.1.1 插入功能流程图 - 8 -](#_Toc121209075)

[3.1.2 插入功能核心代码 - 8 -](#_Toc121209076)

[3.1.3 插入功能截屏示例 - 9 -](#_Toc121209077)

[3.2 按考号删除功能的实现 - 10 -](#_Toc121209078)

[3.2.1 按考号删除功能流程图 - 10 -](#_Toc121209079)

[3.2.2 按考号删除功能核心代码 - 11 -](#_Toc121209080)

[3.2.3 按考号删除功能截屏示例 - 11 -](#_Toc121209081)

[3.3 按考号查找功能的实现 - 12 -](#_Toc121209082)

[3.3.1 按考号查找功能流程图 - 12 -](#_Toc121209083)

[3.3.2 按考号查找功能核心代码 - 12 -](#_Toc121209084)

[3.3.3 按考号查找功能截屏示例 - 13 -](#_Toc121209085)

[3.4 修改功能的实现 - 14 -](#_Toc121209086)

[3.4.1 修改功能流程图 - 14 -](#_Toc121209087)

[3.4.2 修改功能核心代码 - 14 -](#_Toc121209088)

[3.4.3 修改功能截屏示例 - 15 -](#_Toc121209089)

[3.5 统计功能的实现 - 16 -](#_Toc121209090)

[3.5.1 统计功能流程图 - 16 -](#_Toc121209091)

[3.5.2 统计功能核心代码 - 16 -](#_Toc121209092)

[3.5.3 统计功能截屏示例 - 17 -](#_Toc121209093)

[4 测试 - 17 -](#_Toc121209094)

[4.1 边界测试 - 17 -](#_Toc121209095)

[4.1.1 初始化无输入数据 - 17 -](#_Toc121209096)

[4.1.2 删除头结点 - 17 -](#_Toc121209097)

[4.1.3 删除后链表为空 - 18 -](#_Toc121209098)

[4.2 出错测试 - 18 -](#_Toc121209099)

[4.2.1 考生人数错误 - 18 -](#_Toc121209100)

[4.2.2 插入时位置输入错误 - 19 -](#_Toc121209101)

[4.2.3 插入考生的考号已经存在 - 19 -](#_Toc121209102)

[4.2.4 删除考号不存在 - 20 -](#_Toc121209103)

[4.2.5 查找考号不存在 - 20 -](#_Toc121209104)

[4.2.6 修改考生时输入的考号已存在 - 20 -](#_Toc121209105)

1 分析

1.1 背景分析

在学校里，考试是必不可少的一环，而考试报名作为举行考试流程的第一项任务，自然也就十分重要，如果报名过程中出现错误或者混乱，很有可能会影响到考试的正常举行，进而影响到学生利益与学校计划。因此一个良好的考试报名系统对学校和学生来说都是至关重要的，现如今随着学生数量的增多与考试所需数据的日益繁杂，传统的手工管理方法不仅工作量大，还比较容易出错，与学校与学生的高要求出现了矛盾。如何能够快速准确的进行考试信息管理，是目前的考试报名系统需要解决的关键问题。

随着计算机科学技术的高速发展，使用计算机来对考试报名系统进行管理，相较于传统手工管理方法，具有信息容量大，出错率低，管理快速，维护便捷等多种优势。这些优势都可以使得考试报名更有效率，帮助考试能够正常进行，能够保障教学活动的顺利进行。因此，开发一个基于计算机操作的考试报名系统是非常有必要的。

1.2 功能分析

作为一个比较简单的考试报名系统，其基础的功能是能够接收输入的考生信息并且储存下来，同时也能够显示已经保存的考生信息。在此基础上，考试报名系统还具有插入，删除，修改功能，能够在系统里实时更改或添加考生信息，以便考生可以随时更改自己的报名情况；系统还拥有查询功能，可以让管理人员和考生能够迅速查到某位考生的信息；最后，考试报名系统还要能够被正常关闭。删除和查询功能最好用考号和所在位置两种方式均可以实现，减少因为忘记信息而导致操作无法进行的情况发生。

综上所述，一个考试报名系统需要有输入、输出、插入、按考号查询、按位置查询、按考号删除、按位置删除、修改、退出的功能。

2 设计

2.1 数据结构设计

如上功能分析所述，该考试报名系统要求大量增加、删除、修改操作，使用数组，链表等数据结构都可以完成这些操作。但数组在进行这些操作的时候可能会移动整个数组，时间复杂度高，相比较之下，链表在进行增加，删除等操作时非常简便，因此最后选择使用带头结点的链表来维护信息，这样使头结点不再特殊，增加和删除头结点时与处理其它结点方法相同，简化了代码。

2.2 类结构设计

首先提供Student类储存考号姓名等信息，并重载流输入输出运算符方便输入输出。此外链表一般包括两个抽象数据类型——链表结点类（ListNode类）与链表类（List类）。ListNode类用来储存每一位考生信息，List类来将考生信息整合在一起，并提供查询，插入，删除，修改等操作。除此之外，本系统还设计了ExamRegisterSystem类，给用户提供与系统交互的接口。为了增强类的封装性，在数据交互时提供了一系列的get和set函数。同时为了使链表更加具有泛用性，本系统将ListNode类与List类都设计为了模板类。

2.3 类成员与操作设计

### 2.3.1 Student类

Student类存放学生各类信息，提供带参构造函数，并重载了流输入输出运算符用于外接直接输出其内容。一个学生可由一个考试号ID唯一确定，提供get\_id()和set\_id()函数，便于外部查找和修改学生信息。

class Student

{

private:

int id;

String name;

String sex;

int age;

String type;

public:

//默认构造函数

Student() = default;

//复制构造函数

Student(const Student& tar);

//获取学号（用于查找）

int get\_id();

//修改学号

void set\_id(const int& new\_id);

//重载等于运算符

bool operator == (const Student& rhs) const;

//重载流插入、流提取运算符

friend istream& operator>>(istream& is, Student& s);

friend ostream& operator<<(ostream& os, const Student& s);

};

### 2.3.2 ListNode类

ListNode类为链表节点类，采用了类模板提高泛用性。提供默认构造函数、带参构造函数以及与外接数据交流的一系列get与set函数。类内数据为data数据域以及后继节点指针域，存储学生信息以及与后继节点相连。

template <typename T>

class ListNode

{

T \*data;

ListNode<T> \*next;

public:

//默认构造函数

ListNode<T>() = default;

//赋值构造函数

ListNode<T>(const T& src);

//获取数据

const T& get\_data();

//修改数据

void set\_data(const T& src);

friend class List<T>;

};

### 2.3.3 List类

List类为链表类，采用了类模板提高泛用性。提供构造函数新建链表，析构函数释放内存。提供getLinkSize()函数获得链表大小，并提供增删查改的各类函数供外界调用。类内数据为表头指针。

template <typename T>

class List

{

private:

ListNode<T>\* start;

int size = 0;

inline ListNode<T>\* findLastByData(const T& tar);

inline ListNode<T>\* findLastByPos(int pos);

public:

//默认构造函数

List();

//析构函数

~List();

//查找指定位置的结点

ListNode<T>\* Locate(int pos);

//向链表尾部插入结点

inline void insert(const T& s);

//向特定位置插入结点

inline bool insert(const T& s, int pos);

//根据数据内容查找结点

inline ListNode<T>\* findByData(const T& s);

//根据在链表中的位置查找结点

inline ListNode<T>\* findByPos(int tarID);

//修改结点数据

inline void modify(T tar, const T& new\_s);

//删除指定位置的结点

inline void remove(int tarID);

//删除指定内容的结点

inline void remove(const T& tar);

//按序打印链表所有结点

inline void printAll(ostream& os = cout);

//获取链表长度

int get\_list\_size();

};

### 2.3.4 ExamRegisterSystem类

ExamRegisterSystem类为学生管理系统类，采用了类模板提高泛用性。对链表类（List类）进行封装，丰富了交互提示信息，增强了输入错误、边界错误处理性能。类继承于一个存储学生信息的链表，类内设一个学生信息类型的s对象用于新增、修改学生信息等操作。

class ExamRegisterSystem : public List<Student>

{

Student s;

public:

//初始化学生信息

void Init();

//处理请求

void Query();

};

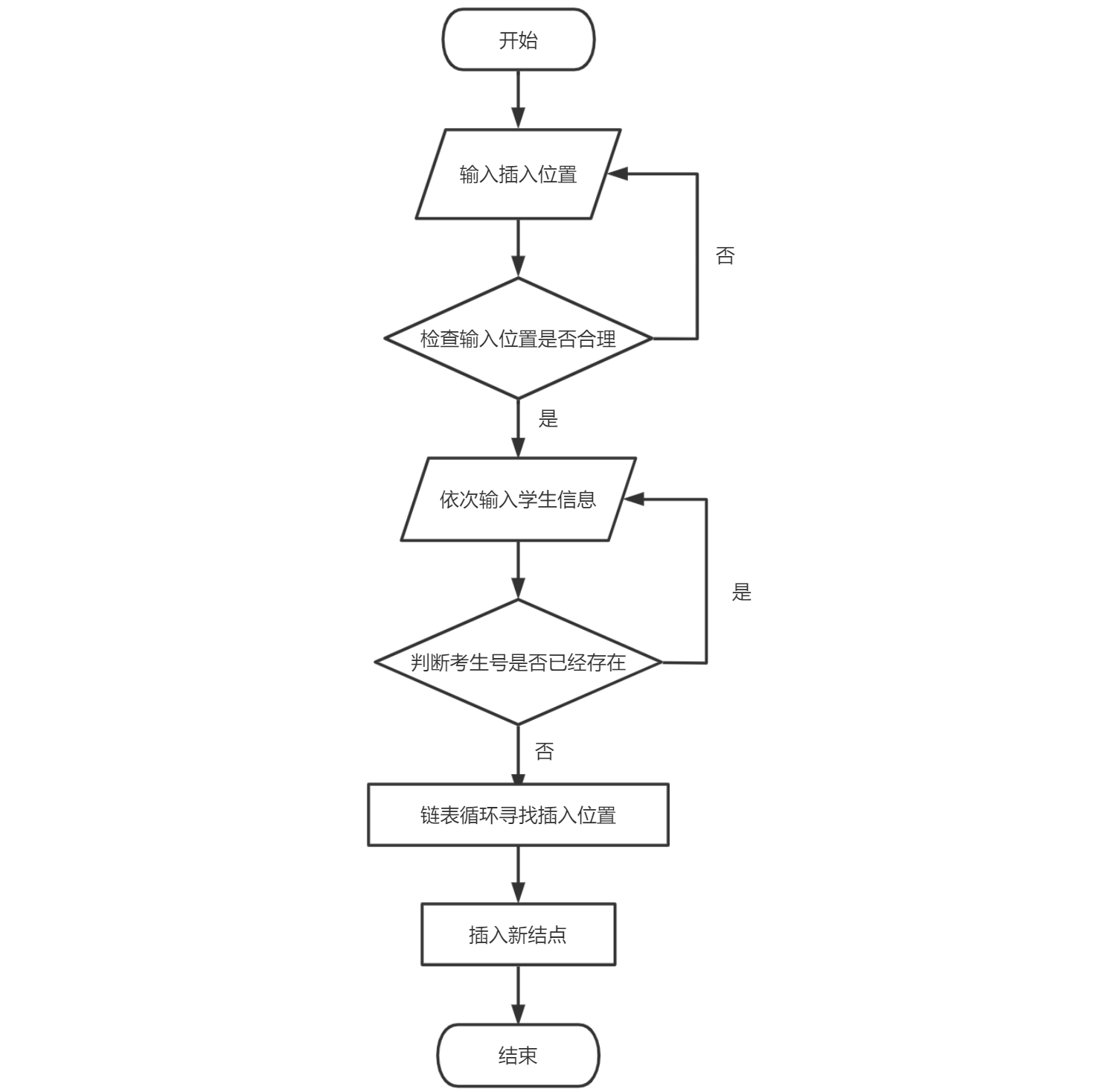
2.4 系统设计

系统在首次使用时，通过调用ExamRegisterSystem类的Init()函数来完成系统的初始化。先让用户输入初始考生录入数量，然后再根据用户后续所输入的操作码完成相应的操作。

# 3 实现

## 3.1 插入功能的实现

3.1.1 插入功能流程图



3.1.2 插入功能核心代码

1.ExamRegisterSystem类

cin >> student\_info;

//输入学号，并判断输入的学号是否有重复

while (findByData(student\_info) != nullptr)

{

cout << "插入失败，该考号已被使用，请重新输入" << endl;

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cin >> student\_info;

}

cout << Prompts[4];

cin >> pos;

//判断位置是否存在，增强程序的鲁棒性

while (pos <= 0 || pos > get\_list\_size() + 1 || cin.fail())

{

cout << "输入非法（插入位置应为正整数，且不超过当前最大位置+1）" << endl;

cout << "请重新输入插入位置：";

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cin >> pos;

}

insert(student\_info, pos);

2.List类中的bool List<T>::insert(const T& s, int pos)函数

template <typename T>

bool List<T>::insert(const T& s, int pos)

{

if (pos <= 0)

{

return false;

}

ListNode<T>\* s1 = findLastByPos(pos);

ListNode<T>\* s2 = new ListNode<T>(s);

if (s2 == nullptr)

{

return false;

}

s2->next = s1->next;

s1->next = s2;

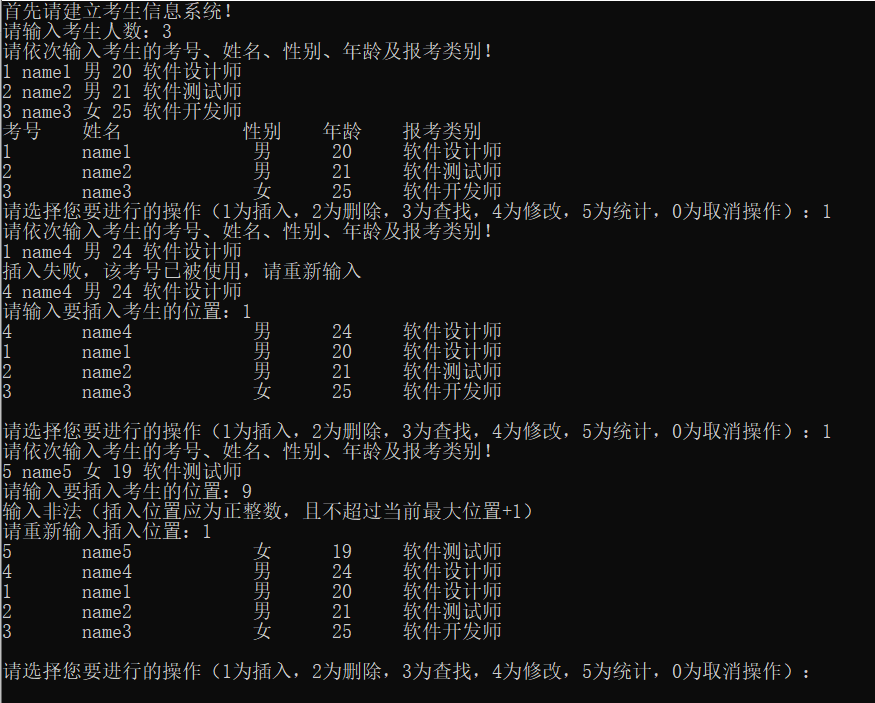
size++;

return true;

}

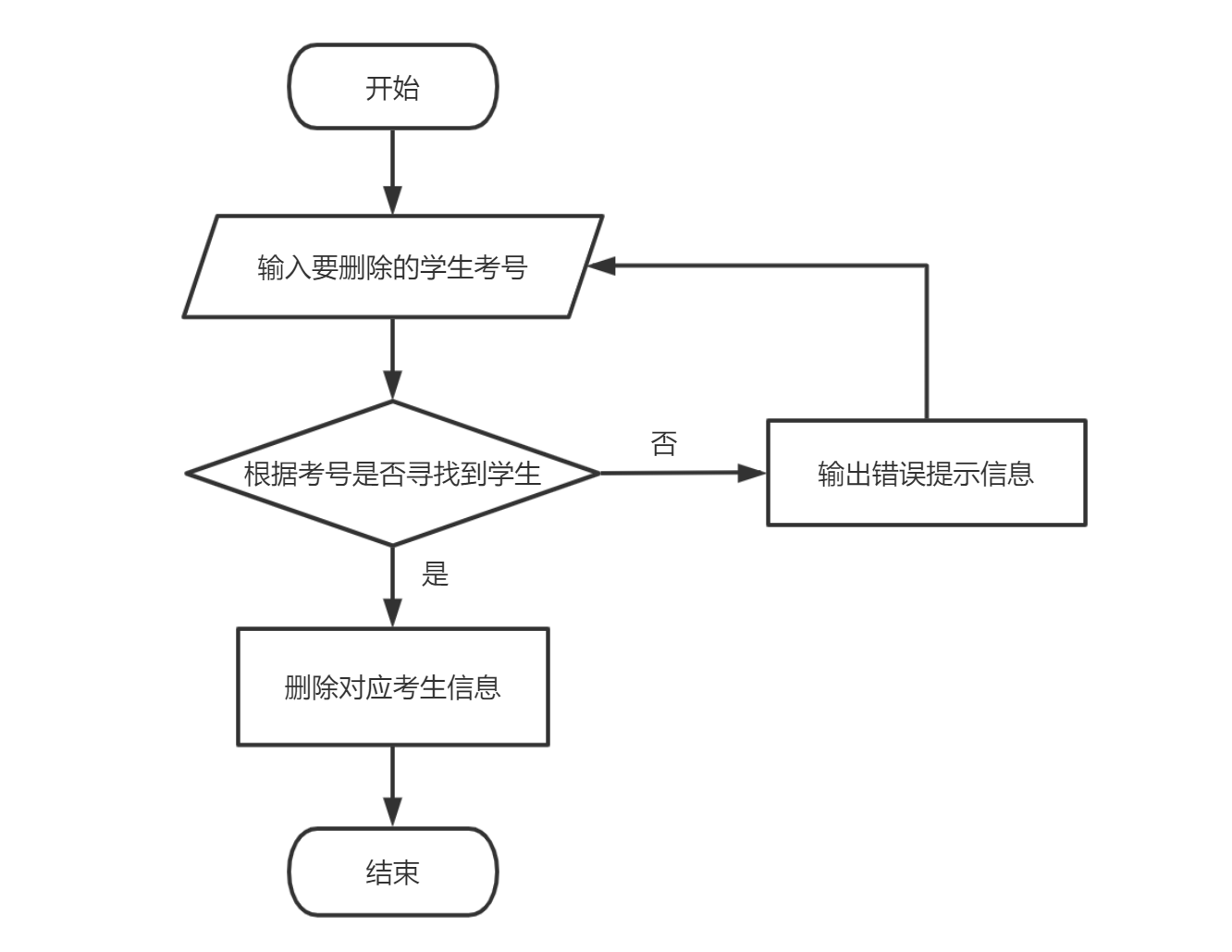
3.1.3 插入功能截屏示例

本示例测试了正常插入、插入位置错误、插入考号已存在时的情况，程序均正确运行。测试结果如下：



## 3.2 按考号删除功能的实现

3.2.1 按考号删除功能流程图



3.2.2 按考号删除功能核心代码

1.ExamRegisterSystem类

cout << Prompts[5];

cin >> ID;

student\_info.set\_id(ID);

//依据学号查找学生

removed\_student = findByData(student\_info);

//未查询到学生，重新输入

while (removed\_student == nullptr)

{

cout << "该考生号对应的考生不存在，请重新输入信息" << endl;

cin >> student\_info;

removed\_student = findByData(student\_info);

}

remove(student\_info);

2.List类中的remove(const Student&tar)函数

template <typename T>

void List<T>::remove(const T& tar)

{

//找到待删除结点的前驱结点

ListNode<T>\* s1 = findLastByData(tar);

ListNode<T>\* tmp = s1->next;

//删除该结点

s1->next = tmp->next;

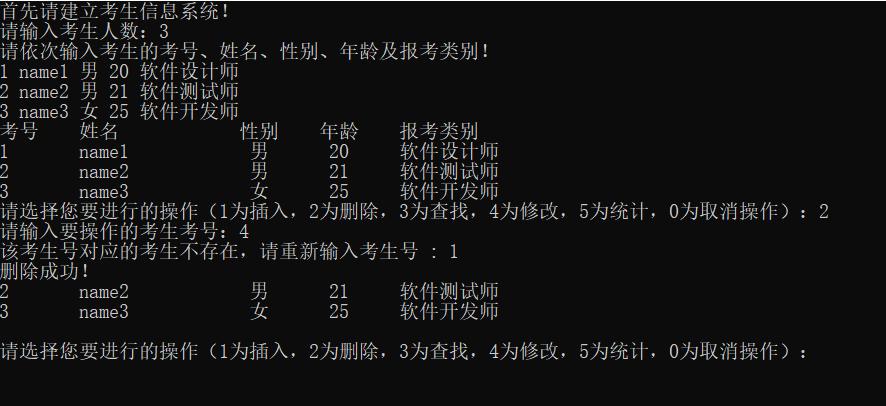
delete tmp;

size--; //链表长度减一

}

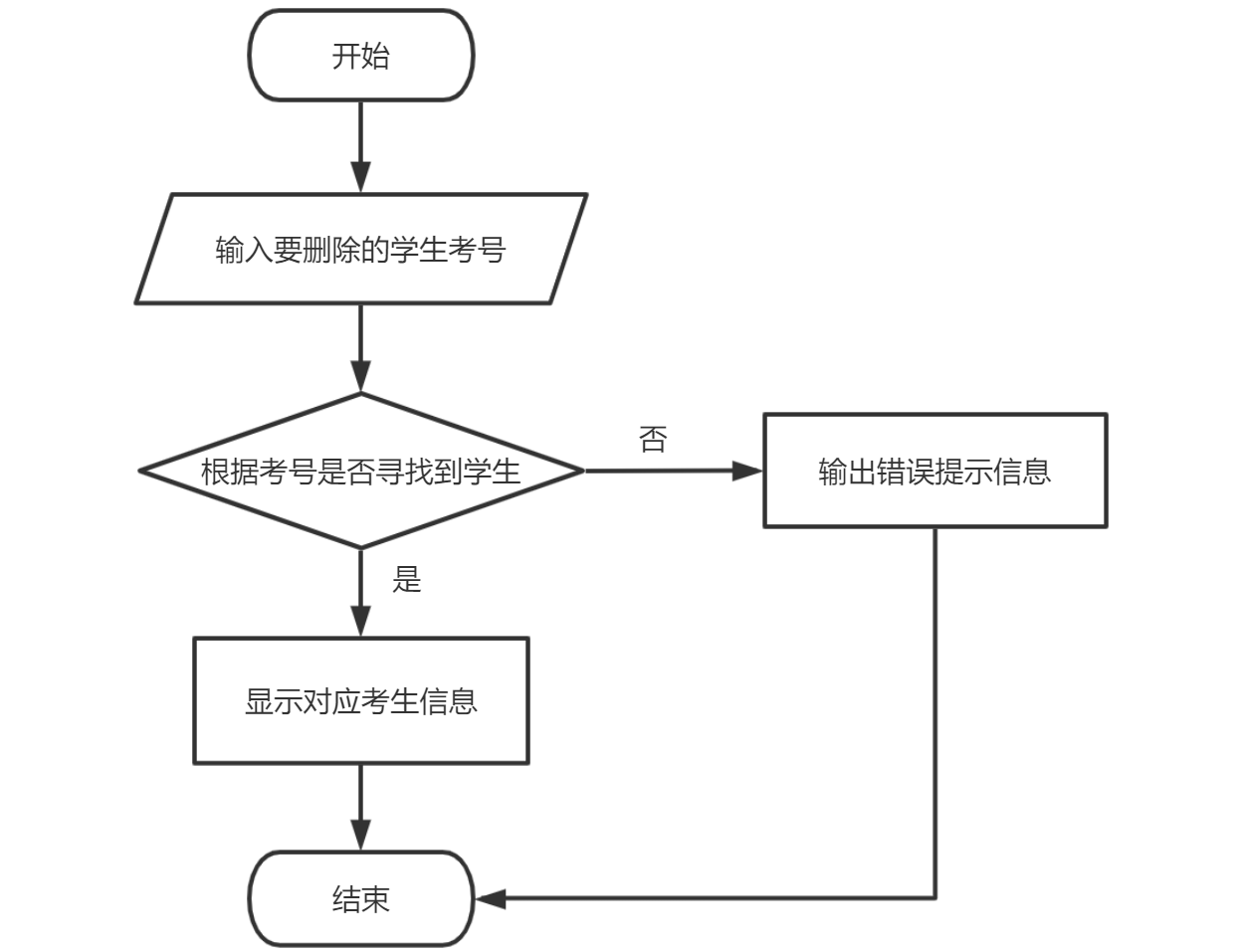
### 3.2.3 按考号删除功能截屏示例

本示例测试了删除已存在考生、删除不存在考生的情况，程序均正确运行。测试结果如下：



## 3.3 按考号查找功能的实现

### 3.3.1 按考号查找功能流程图



### 3.3.2 按考号查找功能核心代码

1.ExamResigterSystem类

cin >> ID;

student\_info.set\_id(ID);

//依据学号查找学生

changed\_student = findByData(student\_info);

//未查询到学生，重新输入

while (changed\_student == nullptr)

{

cout << "该考生号对应的考生不存在，请重新输入考生号" << endl;

cin >> ID;

student\_info.set\_id(ID);

changed\_student = findByData(student\_info);

}

cout << "查询成功，您查询的考生信息为：" << endl;

cout << findByData(student\_info)->get\_data();

2.List类中的findByData(const Student&s)函数

template <typename T>

ListNode<T>\* List<T>::findByData(const T& tar)

{

ListNode<T>\* last\_node = findLastByData(tar);

if (last\_node != nullptr)

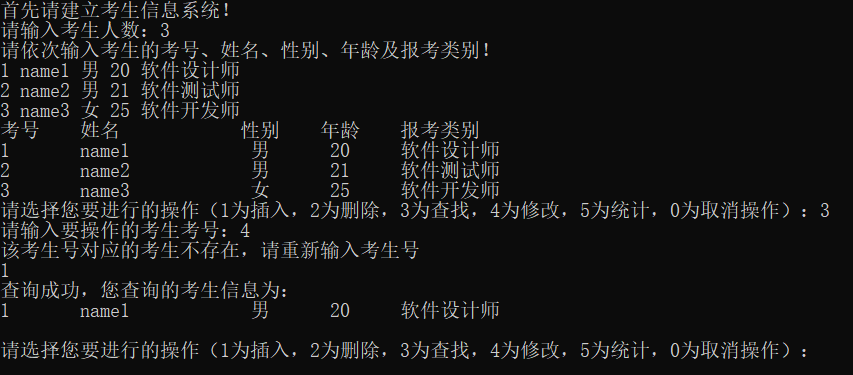
return findLastByData(tar)->next;

return nullptr;

}

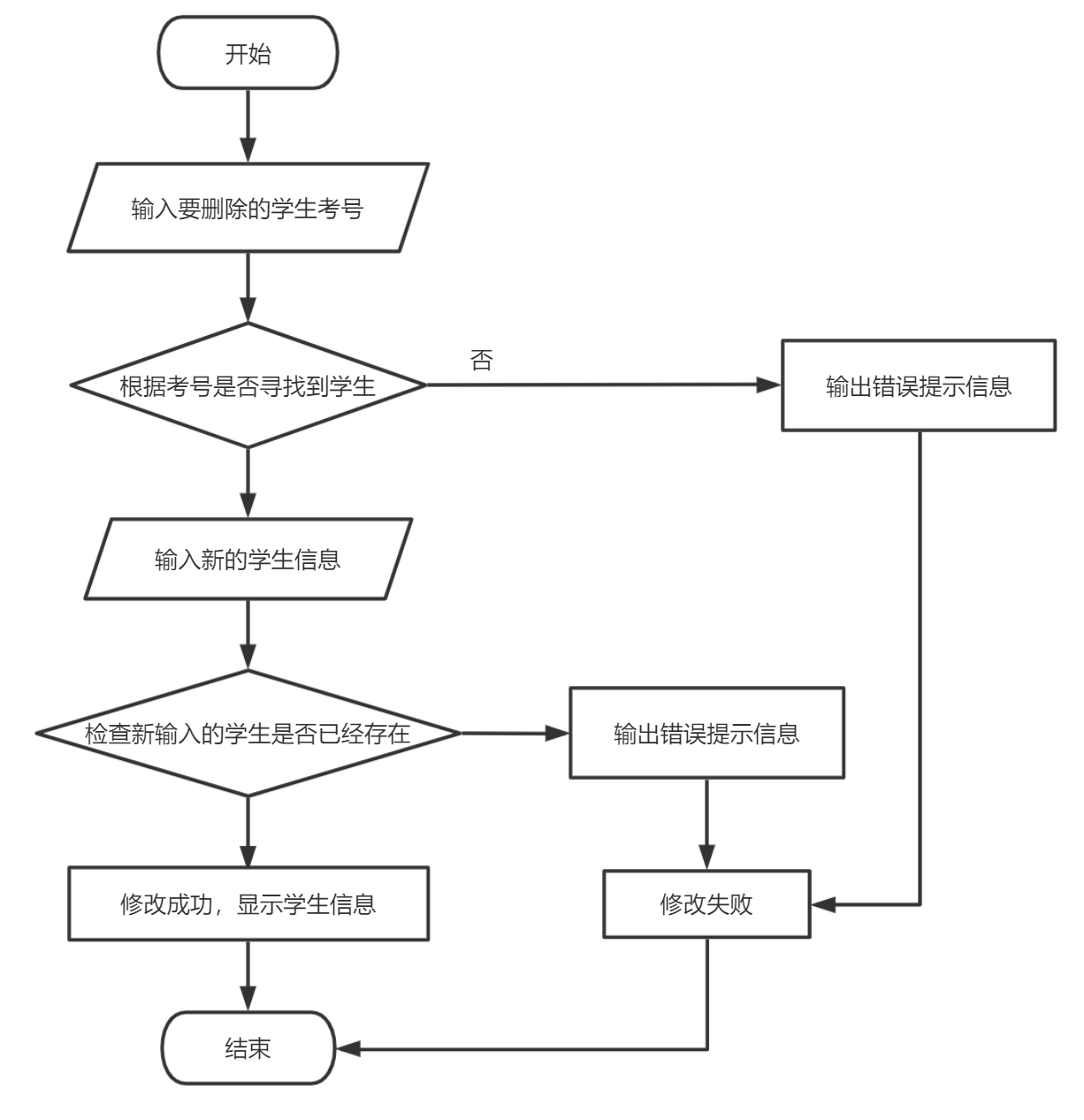
3.3.3 按考号查找功能截屏示例

本示例测试了寻找已存在考生、寻找不存在考生的情况，程序均正确运行。测试结果如下：



3.4 修改功能的实现

3.4.1 修改功能流程图



3.4.2 修改功能核心代码

1.ExamRegisterSystem类

cin >> ID;

student\_info.set\_id(ID);

//依据学号查找学生

changed\_student = findByData(student\_info);

//未查询到学生，重新输入

while (changed\_student == nullptr)

{

cout << "该考生号对应的考生不存在，请重新输入考生号" << endl;

cin >> ID;

student\_info.set\_id(ID);

changed\_student = findByData(student\_info);

}

cout << Prompts[2];

cin >> tmp;

//检查新考号是否存在

exist\_info = findByData(tmp);

while (exist\_info != nullptr)

{

cout << "修改失败，该考号已经存在，请重新输入信息" << endl;

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cin >> tmp;

exist\_info = findByData(tmp);

}

modify(student\_info, tmp);

2.List类中的modify(Student tar, const Student& new\_s)函数

template <typename T>

void List<T>::modify(T tar, const T& new\_s)

{

ListNode<T>\* s1 = findByData(tar);

if (s1 != nullptr)

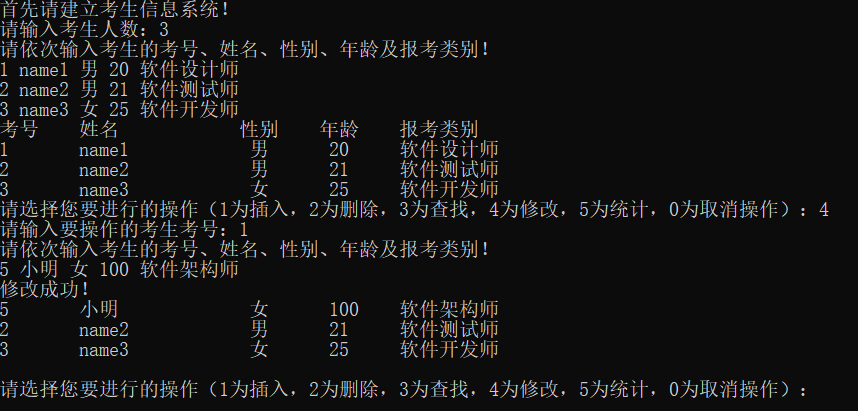
{

\*s1->data = new\_s;

}

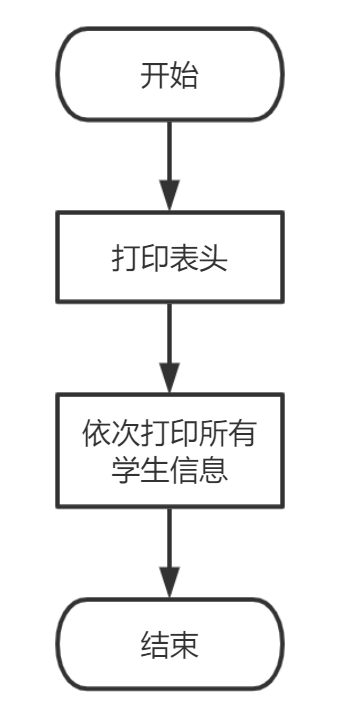
}

3.4.3 修改功能截屏示例



3.5 统计功能的实现

3.5.1 统计功能流程图



3.5.2 统计功能核心代码

List类中的printAll()函数

template <typename T>

void List<T>::printAll(ostream& os)

{

ListNode<T>\* s = start->next;

while (s != nullptr)

{

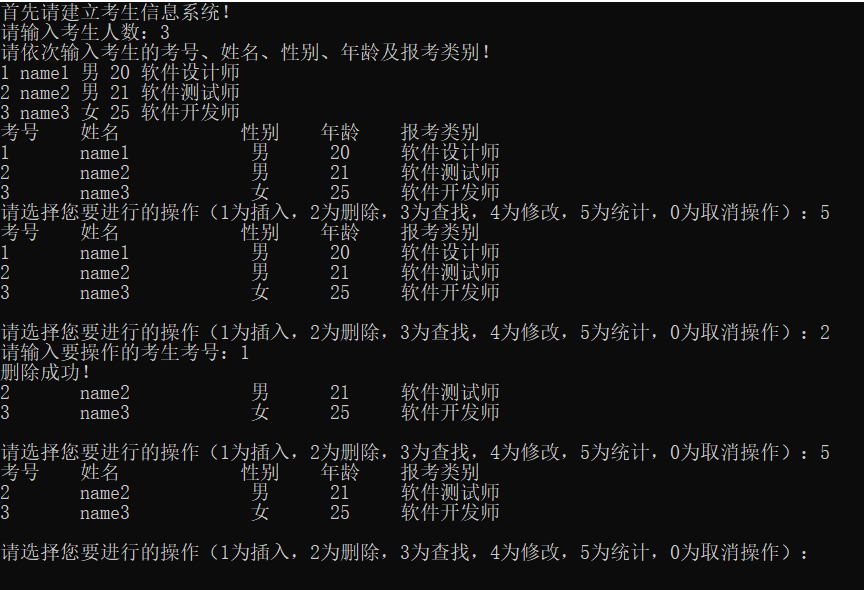
os << \*s->data;

s = s->next;

}

}

3.5.3 统计功能截屏示例



4 测试

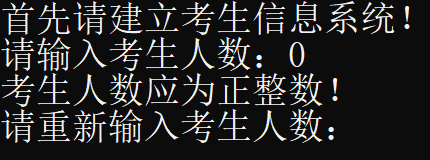
4.1 边界测试

4.1.1 初始化无输入数据

**测试用例：**初始无输入数据

**预期结果：**给出错误提示，程序运行正常不崩溃。

**实验结果：**

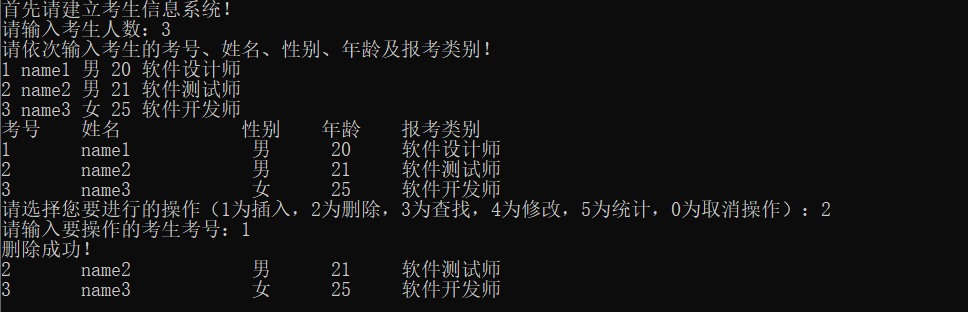


4.1.2 删除头结点

**测试用例：**删除头结点

**预期结果：**程序正常运行，不崩溃。

**实验结果：**

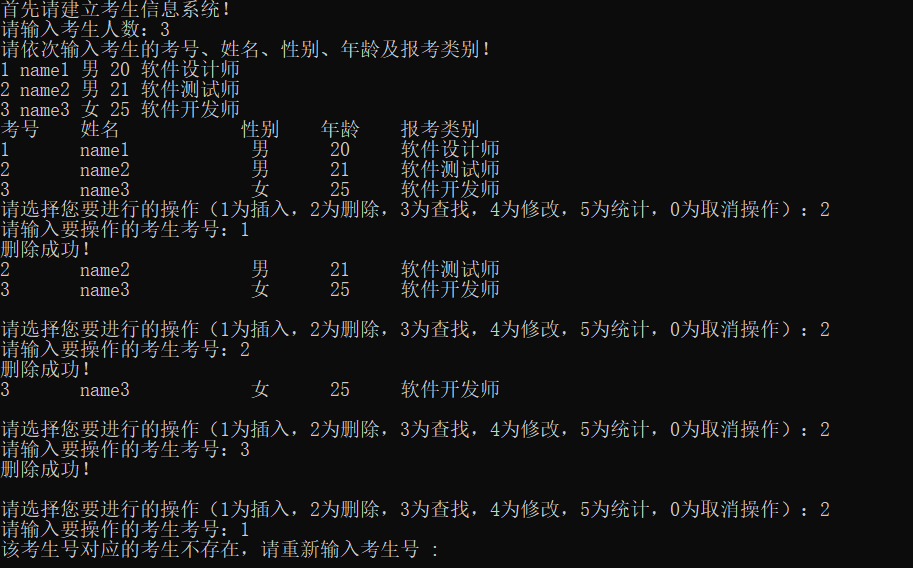


4.1.3 删除后链表为空

**测试用例：**删除前链表只有一个结点，删除后链表为空

**预期结果：**程序正常运行，不崩溃。

**实验结果：**



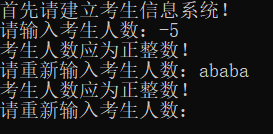
4.2 出错测试

4.2.1 考生人数错误

**测试用例：**输入考生人数为负数或者不是数字

**预期结果：**程序给出提示信息，程序正常运行不崩溃。

**实验结果：**

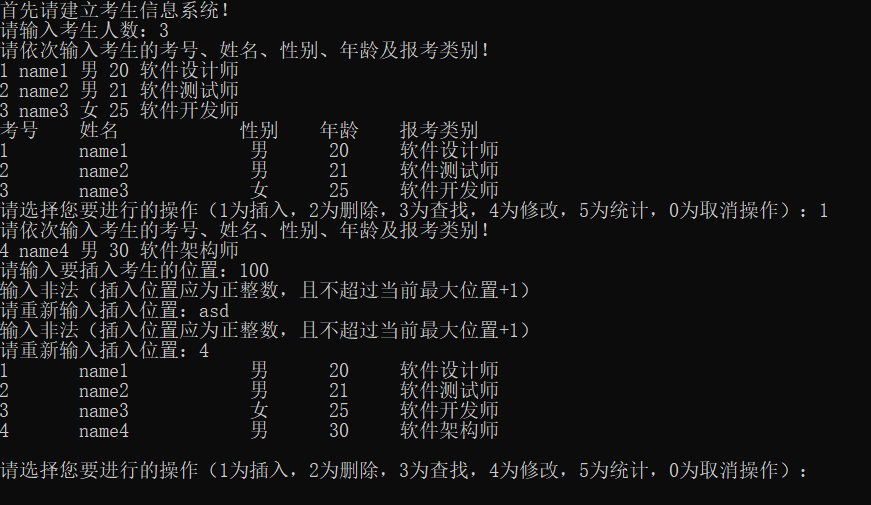


4.2.2 插入时位置输入错误

**测试用例：**插入时输入位置不存在或不为数字

**预期结果：**程序给出提示信息，程序正常运行不崩溃。

**实验结果：**

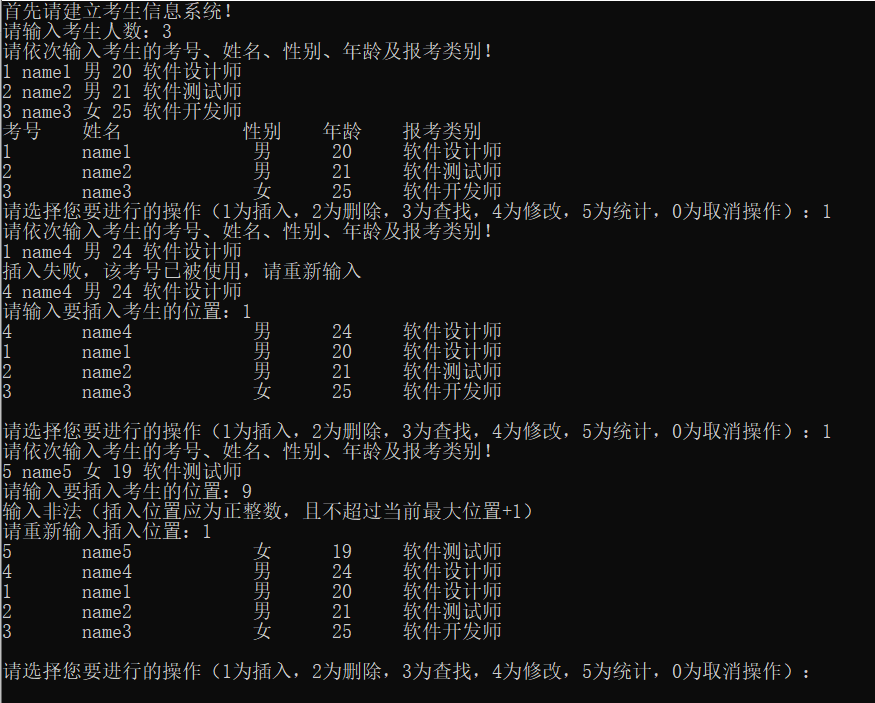


4.2.3 插入考生的考号已经存在

**测试用例：**系统中有三条信息，向其中插入考号已经存在于系统中的考生

**预期结果：**程序给出提示信息，程序正常运行不崩溃。

**实验结果：**

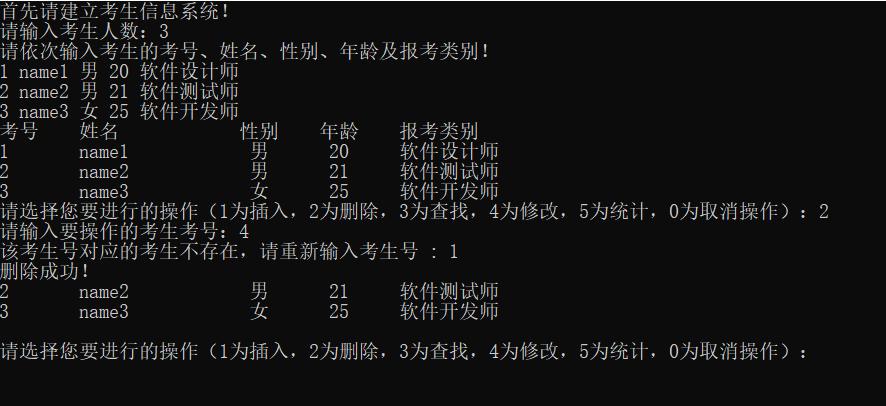


4.2.4 删除考号不存在

**测试用例：**在按考号删除时，要删除的考号不存在

**预期结果：**程序给出提示信息，程序正常运行不崩溃。

**实验结果：**

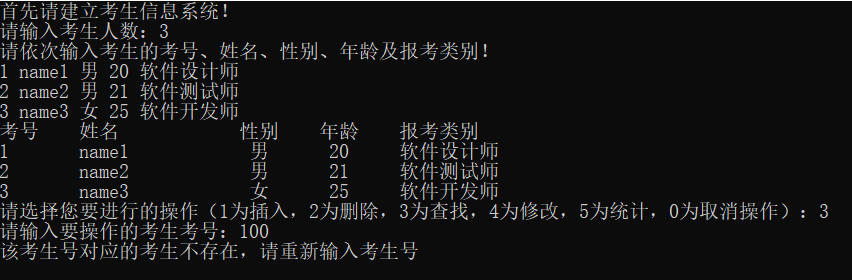


4.2.5 查找考号不存在

**测试用例：**在按考号查找时，要查找的考号不存在

**预期结果：**程序给出提示信息，程序正常运行不崩溃。

**实验结果：**



4.2.6 修改考生时输入的考号已存在

**测试用例：**系统中有三条信息，修改考生的考号为系统中存在的考号

**预期结果：**程序给出提示信息，程序正常运行不崩溃。

**实验结果：**

