项目说明文档

数据结构课程设计

——两个有序链表序列的交集

作 者 姓 名： 祝新元

学 号： 1751629

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 1](#_Toc495668153)

[1.1 背景分析 1](#_Toc495668154)

[1.2 功能分析 1](#_Toc495668155)

[2 设计 2](#_Toc495668156)

[2.1 数据结构设计 2](#_Toc495668157)

[2.2 类结构设计 2](#_Toc495668158)

[2.3 成员与操作设计 2](#_Toc495668159)

[2.4 系统设计 4](#_Toc495668160)

[3 实现 4](#_Toc495668161)

[3.1 插入功能的实现 4](#_Toc495668162)

[3.1.1 插入功能流程图 4](#_Toc495668163)

[3.1.2 插入功能核心代码 5](#_Toc495668164)

[3.1.3 插入功能截屏示例 5](#_Toc495668165)

[3.2 删除功能的实现 7](#_Toc495668166)

[3.2.1 删除功能流程图 7](#_Toc495668167)

[3.2.2 删除功能核心代码 7](#_Toc495668168)

[3.2.3 删除功能截屏示例 8](#_Toc495668169)

[3.3 查找功能的实现 10](#_Toc495668170)

[3.3.1 查找功能流程图 10](#_Toc495668171)

[3.3.2 查找功能核心代码 10](#_Toc495668172)

[3.3.3 查找功能截图示例 11](#_Toc495668173)

[3.4 修改功能的实现 13](#_Toc495668174)

[3.4.1 修改功能流程图 13](#_Toc495668175)

[3.4.2 修改功能核心代码 13](#_Toc495668176)

[3.4.3 修改功能截屏示例 14](#_Toc495668177)

[3.5 统计功能的实现 15](#_Toc495668178)

[3.5.1 统计功能流程图 15](#_Toc495668179)

[3.5.2 统计功能核心代码 15](#_Toc495668180)

[3.5.3 统计功能截屏示例 16](#_Toc495668181)

[3.6 总体系统的实现 16](#_Toc495668182)

[3.6.1 总体系统流程图 16](#_Toc495668183)

[3.6.2 总体系统核心代码 17](#_Toc495668184)

[3.6.3 总体系统截屏示例 17](#_Toc495668185)

[4 测试 19](#_Toc495668186)

[4.1 功能测试 19](#_Toc495668187)

[4.1.1 插入功能测试 19](#_Toc495668188)

[4.1.2 删除功能测试 19](#_Toc495668189)

[4.1.3 查找功能测试 20](#_Toc495668190)

[4.1.4 修改功能测试 20](#_Toc495668191)

[4.1.5 统计功能测试 21](#_Toc495668192)

[4.2 边界测试 21](#_Toc495668193)

[4.2.1 初始化无输入数据 21](#_Toc495668194)

[4.2.2 删除头结点 22](#_Toc495668195)

[4.2.3 删除后链表为空 22](#_Toc495668196)

[4.3 出错测试 23](#_Toc495668197)

[4.3.1 考生人数错误 23](#_Toc495668198)

[4.3.2 操作码错误 23](#_Toc495668199)

[4.3.3 插入位置不存在 23](#_Toc495668200)

[4.3.4 删除考号不存在 24](#_Toc495668201)

[4.3.5 查找考号不存在 24](#_Toc495668202)

[4.3.6 修改考号不存在 25](#_Toc495668203)

# 1 分析

## 1.1 背景分析

已知两个非降序链表序列S1和S2，设计函数构造出S1和S2的交集新链表S3。

## 1.2 功能分析

1. 输入说明：输入分2行，分别在每行给出由若干个正整数构成的非降序序列，用-1表示序列的结尾（-1不属于这个序列）。数字用空格间隔。
2. 输出说明：在一行中输出两个输入序列的交集序列，数字间用空格分开，结尾不能有多余空格；若新链表为空，输出NULL。

# 2 设计

## 2.1 数据结构设计

如上功能分析所述，该系统要求大量的增加、删除、修改操作，而链表进行增加、删除等操作十分简便，因此考虑使用链表数据结构。同时，为了实现简易，在第一个结点之前附加一个头结点，这样就使得增加或者删除头结点与处理其他结点方法相同，使得程序简洁。

## 2.2 类结构设计

经典的链表一般包括两个抽象数据类型（ADT）——链表结点类（LNode）与链表类（LinkList），而两个类之间的耦合关系可以采用嵌套、继承等多种关系。为方便处理，本系统采用struct描述链表结点类（LNode），这样使得链表结点类（LinkList）可以访问链表结点。

## 2.3 成员与操作设计

**链表结点类（LNode）**

**公有成员：**

string m\_studentID; //表示学生考号

string m\_studentName;//表示学生姓名

string m\_sex;//表示学生性别

string m\_age;//表示学生年龄

string m\_category;//表示学生报考种类

LNode \*m\_next;//链表结点的指针域

**公有操作：**

LNode(string studentID = "",

string studentName = "",

string sex = "",

string age = "",

string category = "",

LNode \*next = nullptr):

m\_studentID(studentID),

m\_studentName(studentName),

m\_sex(sex),

m\_age(age), m\_category(category),

m\_next(next)

{}//具有默认参数的构造函数

**链表类（LinkList）**

**私有成员：**

LNode \*m\_first;//链表的头指针

LNode \*m\_length;//链表的长度

**公有操作：**

LinkList::LinkList();

//LinkList的构造函数，在堆内存开辟一个附加头结点并使m\_first指向附加头结点

LinkList::~LinkList();

//LinkList的析构函数，通过调用MakeEmpty()实现对内存的回收

void LinkList::InputData(int studentNumber)

//采用后插法建立链表，输入studentNumber个数据

void LinkList::MakeEmpty();

//收回链表所占的内存，防止内存泄漏

void LinkList::Display() const;

//显示链表所存储的数据

void LinkList::Insert()

//在链表中插入一个结点

void LinkList::Delete()

//在链表中删除一个结点

void LinkList::Find() const

//在链表中查找一个结点

void LinkList::Change()

//在链表中修改一个结点

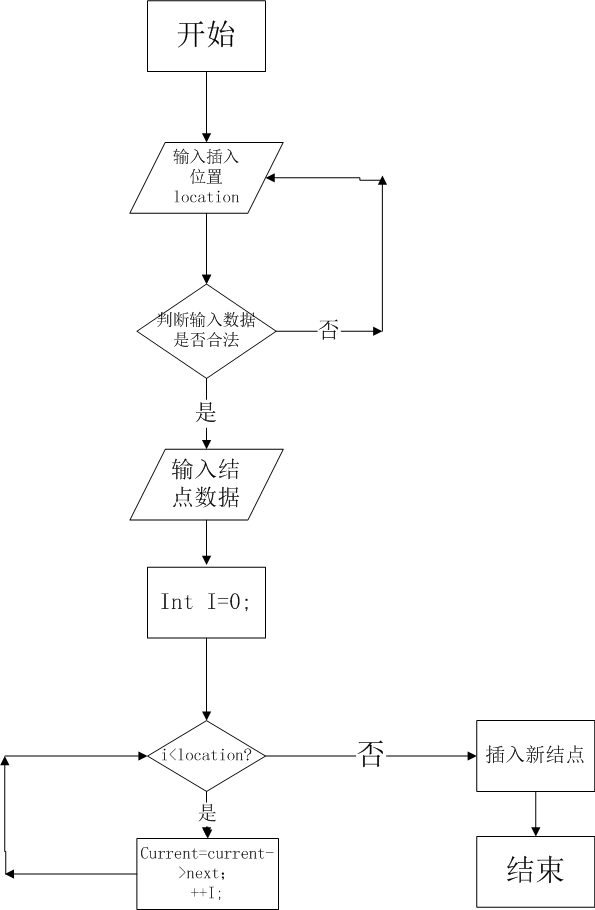
## 2.4 系统设计

系统首先调用InitScreen()函数实现对屏幕的初始化，完成对链表L的创建和输入数据工作，然后根据用户所输入的操作码（operatorCode）执行链表L对应的成员函数。

# 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

LNode \*temp = new LNode(studentID, studentName, sex, age, category);

LNode \*current = m\_first;

//寻找插入位置前一个节点

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

{

current = current->m\_next;

}

//进行插入操作

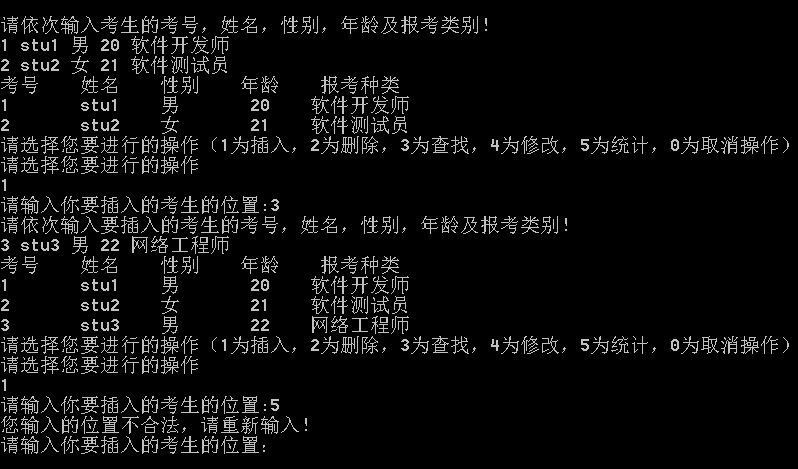
++m\_length;

temp->m\_next = current->m\_next;

current->m\_next = temp;

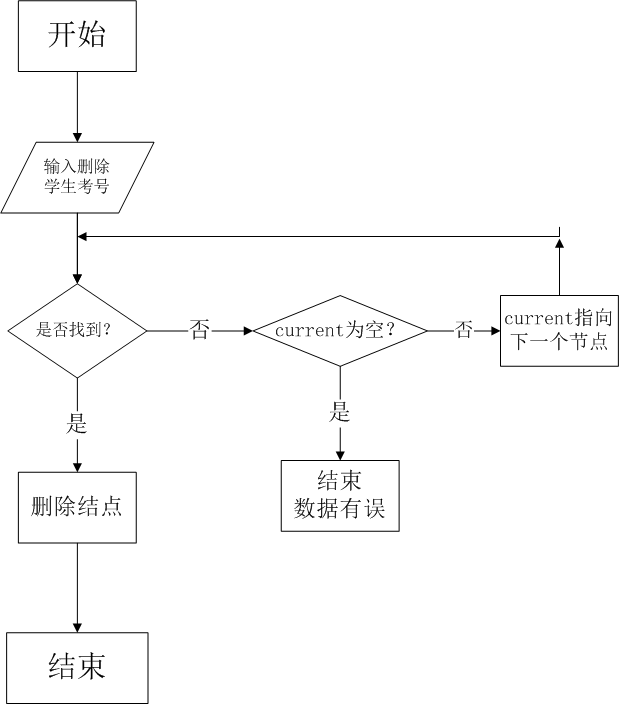
### 3.1.3 插入功能截屏示例





## 3.2 删除功能的实现

### 3.2.1 删除功能流程图



### 3.2.2 删除功能核心代码

bool LinkList::Delete()

{

LNode \*current = m\_first->m\_next;//要删除的节点

LNode \*prior = m\_first;//要删除节点的前一个节点

int flag = 0; //找到的标志

while ((flag == 0) && (current != nullptr))

{

if (current->m\_studentID == studentID)

{

flag = 1;

}

else

{

current = current->m\_next;

prior = prior->m\_next;

}

}

//进行删除操作

{

--m\_length;

cout << "你删除的考生的信息是："

<< current->m\_studentID << " "

<< current->m\_studentName << " "

<< current->m\_sex << " "

<< current->m\_age << " "

<< current->m\_category << " "

<< endl;

prior->m\_next = current->m\_next;

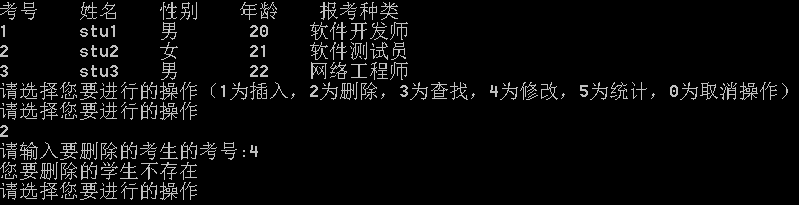
delete current;

}

}

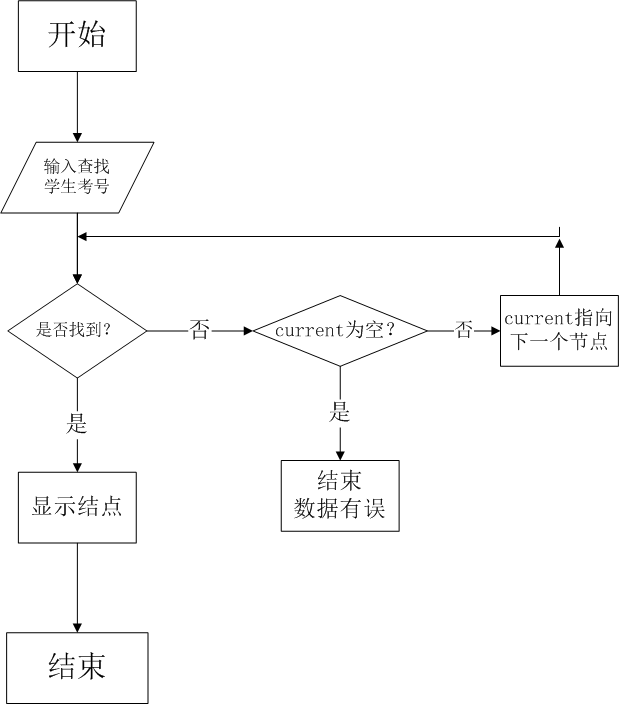
### 3.2.3 删除功能截屏示例





## 3.3 查找功能的实现

### 3.3.1 查找功能流程图



### 3.3.2 查找功能核心代码

bool LinkList::Find() const

{

LNode \*current = m\_first->m\_next;//当前寻找的节点

int flag = 0; //找到的标志

while ((flag == 0) && (current != nullptr))

{

if (current->m\_studentID == studentID)

{

flag = 1;

}

else

{

current = current->m\_next;

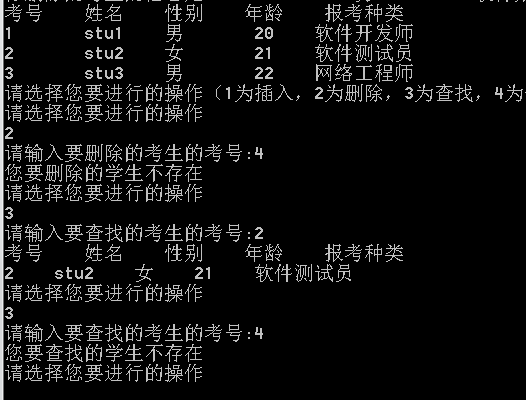
}

}

}

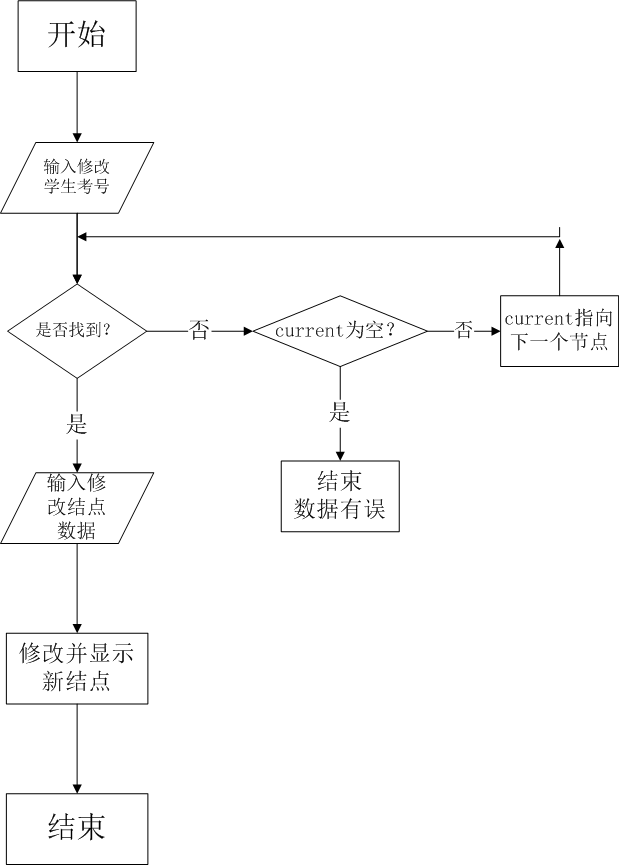
### 3.3.3 查找功能截图示例





## 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

bool LinkList::Change()

{

LNode \*current = m\_first->m\_next;//当前修改的节点

int flag = 0; //找到的标志

while ((flag == 0) && (current != nullptr))

{

if (current->m\_studentID == studentID)

{

flag = 1;

}

else

{

current = current->m\_next;

}

}

{

current->m\_studentID = studentID;

current->m\_studentName = studentName;

current->m\_sex = sex;

current->m\_age = age;

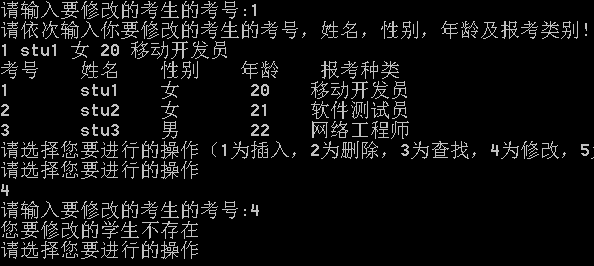
current->m\_category = category;

}

}

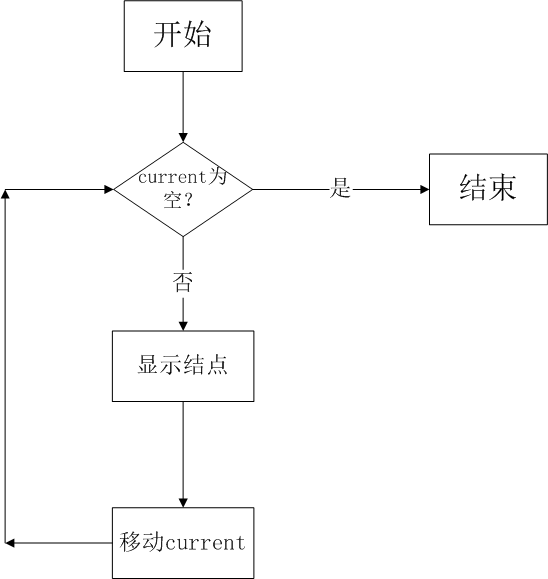
### 3.4.3 修改功能截屏示例





## 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

while (current != nullptr)

{

if (current->m\_studentID != "")

{

cout << current->m\_studentID << " "

<< current->m\_studentName << " "

<< current->m\_sex << " "

<< current->m\_age << " "

<< current->m\_category << " "

<< endl;

}

current = current->m\_next;//移动到下一个节点

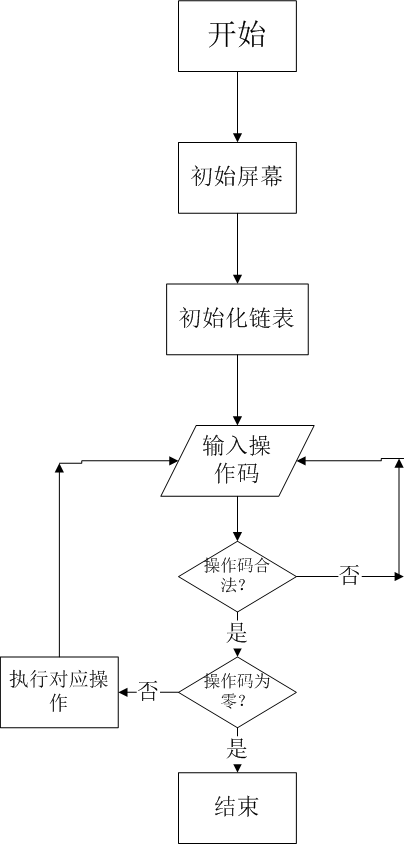
}

### 3.5.3 统计功能截屏示例



## 3.6 总体系统的实现

### 3.6.1 总体系统流程图



### 3.6.2 总体系统核心代码

while (operateCode != '0')

{

switch (operateCode)

{

case '1': L.Insert(); break;

case '2': L.Delete(); break;

case '3': L.Find(); break;

case '4': L.Change(); break;

case '5': L.Display(); break;

//如果用户输入操作要求错误，提示用户重新输入

default: cout << "抱歉，您想要进行的操作不存在，请重新输入！" << endl; break;

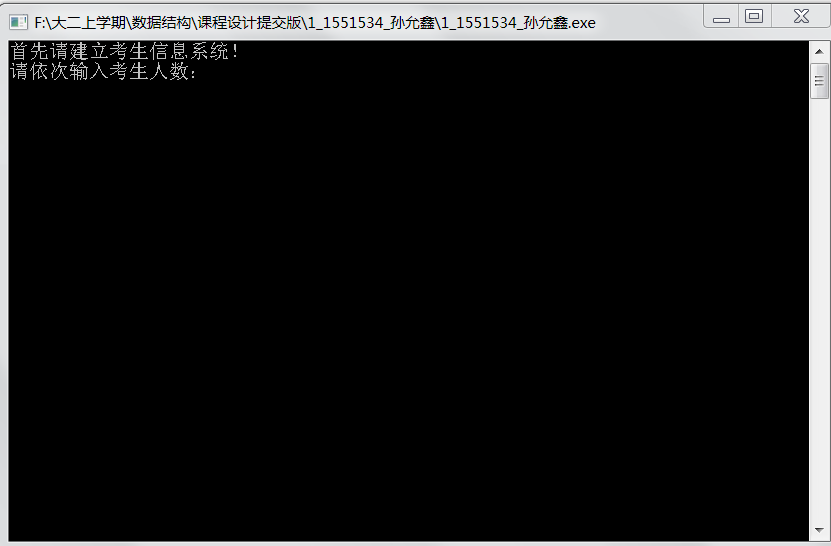
}

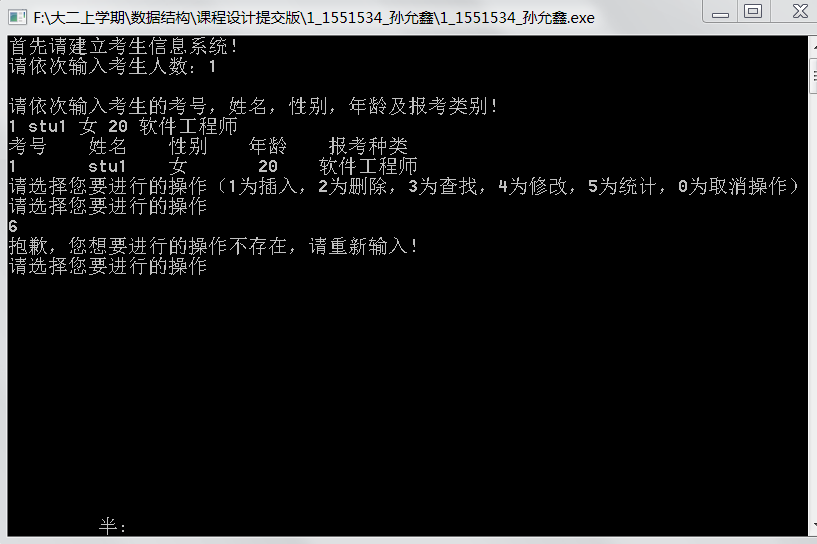
cout << "请选择您要进行的操作" << endl;

cin >> operateCode;

}

### 3.6.3 总体系统截屏示例





# 4 测试

## 4.1 功能测试

### 4.1.1 插入功能测试

**测试用例**：3 stu3 男 22 网络工程师

**预期结果**：

1 stu1 男 20 软件开发师

2 stu2 女 21 软件测试员

3 stu3 男 22 网络工程师

**实验结果**



### 4.1.2 删除功能测试

**测试用例：**删除考号为4的考生

**预期结果：**

1 stu1 男 20 软件开发师

2 stu2 女 21 软件测试员

3 stu3 男 22 网络工程师

**实验结果：**



### 4.1.3 查找功能测试

**测试用例：**查找考号为2的考生

**预期结果：**

2 stu2 女 21 软件测试员

**实验结果：**



### 4.1.4 修改功能测试

**测试用例：**将考号1修改为性别女，年龄20，报考种类移动开发员。

**预期结果：**

1 stu1 女 20 移动开发员

**实验结果：**



### 4.1.5 统计功能测试

**测试用例：**统计当前数据

**预期结果：**

1 stu1 女 20 移动开发员

2 stu2 女 21 软件测试员

3 stu3 男 22 网络工程师

**实验结果：**



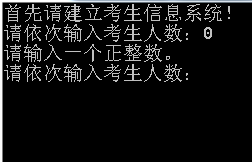
## 4.2 边界测试

### 4.2.1 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：**

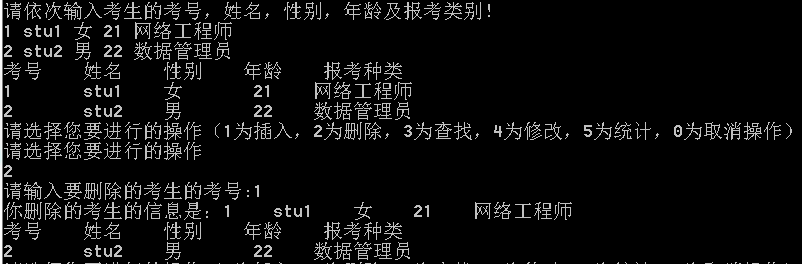


### 4.2.2 删除头结点

**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：**

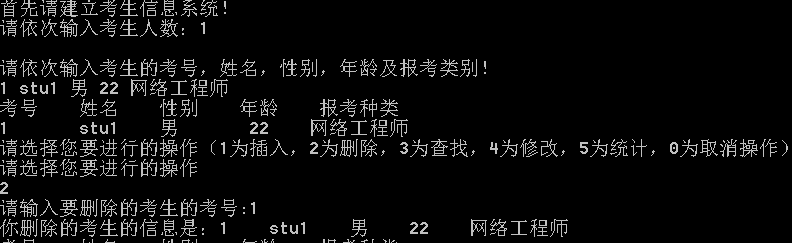


### 4.2.3 删除后链表为空

**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：**



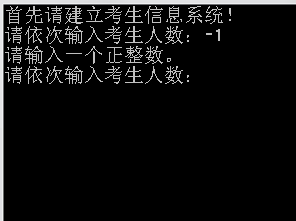
## 4.3 出错测试

### 4.3.1 考生人数错误

**测试用例：**输入考生人数为负数

**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：**

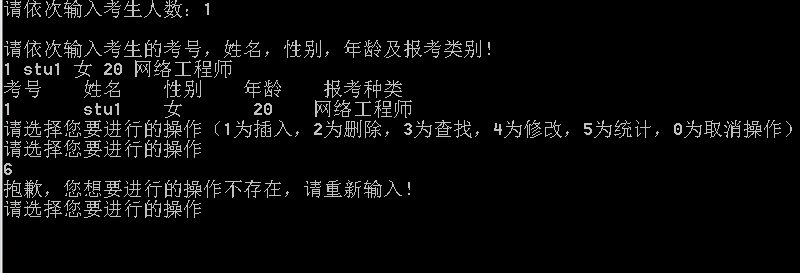
****

### 4.3.2 操作码错误

**测试用例：**输入操作码错误

**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：**

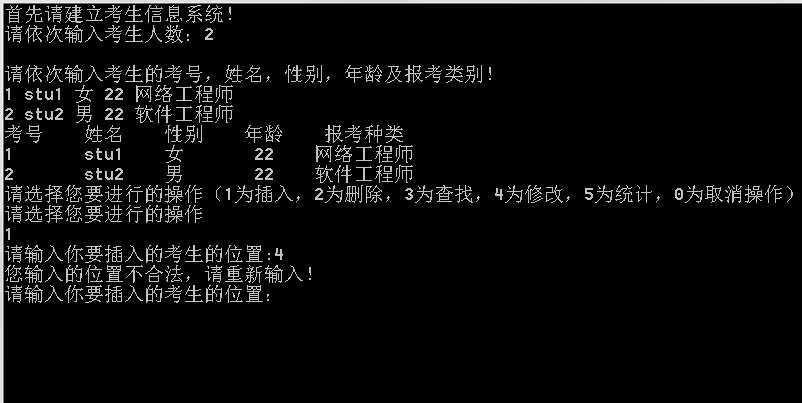
****

### 4.3.3 插入位置不存在

**测试用例：**链表里有两条记录，向链表的第四个位置插入结点

**预期结果：**程序给出错误信息，程序正常运行不崩溃。

**实验结果：**

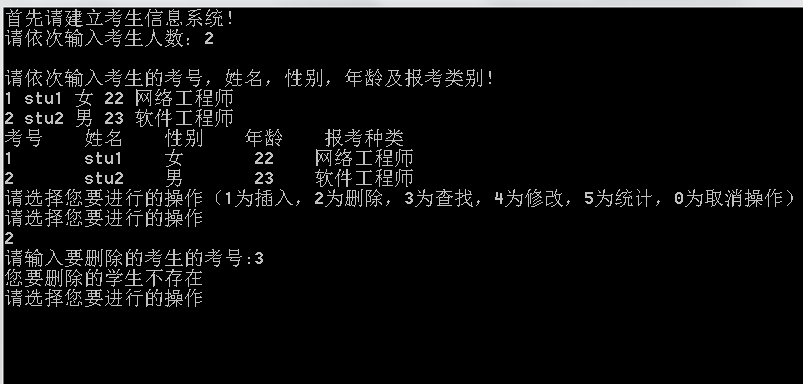


### 4.3.4 删除考号不存在

**测试用例：**要删除的考号不存在

**预期结果：**程序给出错误信息，程序正常运行不崩溃。

**实验结果：**

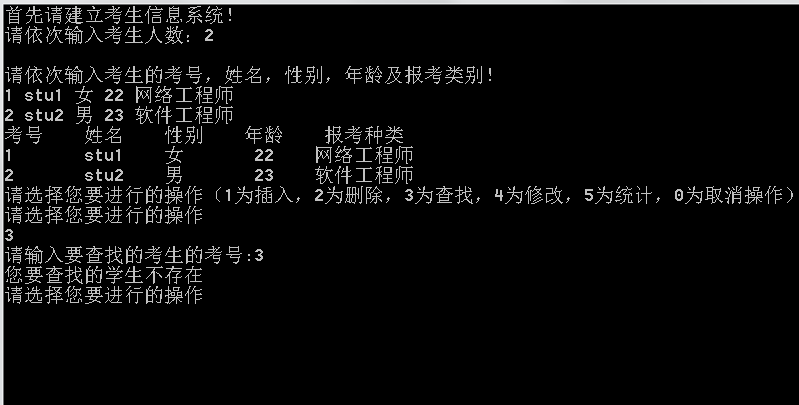
****

### 4.3.5 查找考号不存在

**测试用例：**要查找的考号不存在

**预期结果：**程序给出错误信息，程序正常运行不崩溃。

**实验结果：**

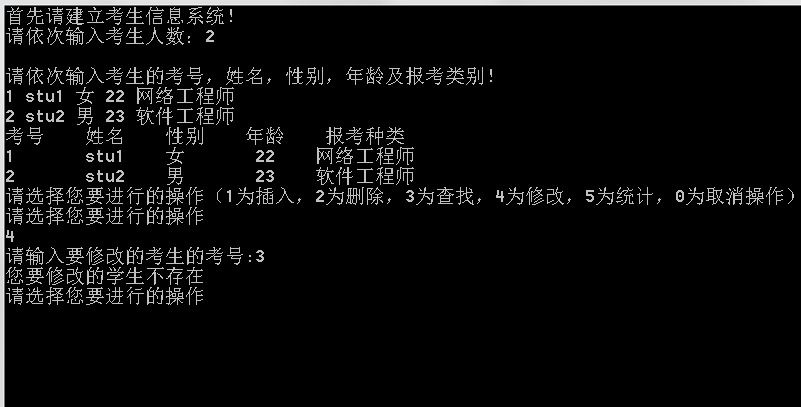
****

### 4.3.6 修改考号不存在

**测试用例：**要修改的考号不存在

**预期结果：**程序给出错误信息，程序正常运行不崩溃。

**实验结果：**



# 