# 1 需求

## 1.1 项目背景

考试报名工作给各高校带来了新的挑战，给教务管理部门增加了很大的工作量。在互联网未普及之前，各机构对考生信息的登记主要依赖人工，不仅效率较低，而且容易出错。而步入信息时代，许多传统的管理方式都实现了信息化、系统化，大量繁琐、重复的操作都可以用程序代替，考试报名管理系统就是如此。本项目旨在用C++简易模拟一个功能相对完整的考试报名管理系统，使其具有基本的“增、删、改、查、统计”功能，完成对考生数据的高效处理。

## 1.2 功能分析

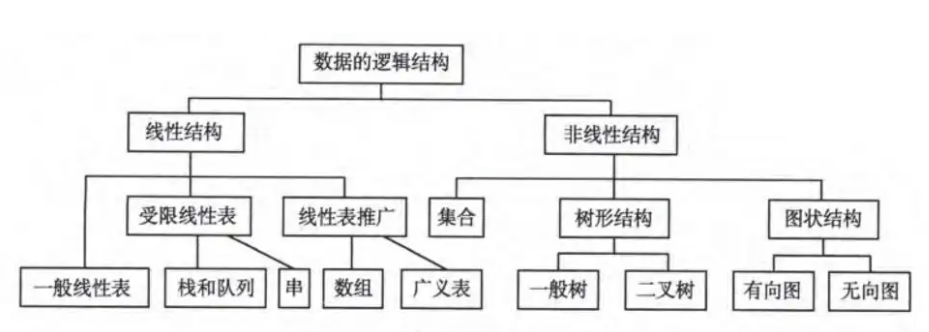
本项目的实质是完成对考生信息的建立，查找，插入，修改，删除，统计等功能。其中考生信息包括准考证号，姓名，性别，年龄和报考类别等信息。本项目在设计时选用链表作为主要的数据结构，定义链表节点类与链表类，通过实现各成员函数以完成对数据操作的相应功能，最后在主函数内调用循环选择函数测试各项操作。其中程序对用户的每次输入进行错误处理，当用户输入错误时会给出提示并让用户再次输入。

# 2 设计

## 2.1 数据结构设计

### 2.1.1 逻辑结构设计

本项目的核心功能是完成对考生信息的增、删、改、查操作。由于每个考生信息相对独立，各个元素之间不存在明显的偏序关系或层次关系，也不存在“一对多”与“多对一”的映射，所以可以采用线性结构进行存储。



### 2.1.2 存储结构设计

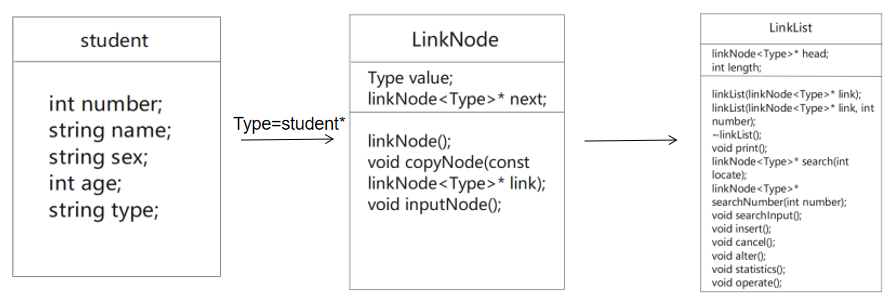
线性表的存储通常通过顺序存储和链式存储方式实现。其中顺序存储在修改操作上的时间复杂度为O(1)，而其他操作均为O(n)，而链式存储在插入、查找、修改等操作上的时间复杂度都是O(1)，综合比较发现在本项目中链式存储的效率明显高于顺序存储，所以本项目采用链式存储方式即链表数据结构存储考生信息。

|  |  |  |  |
| --- | --- | --- | --- |
|  | 顺序存储 | 链式存储 | 比较结果 |
| 插入功能 | O(n) | O(1) | 链式存储更优 |
| 删除功能 | O(n) | O(1) | 链式存储更优 |
| 查找功能 | O(n) | O(n) | 相同 |
| 修改功能 | O(1) | O(1) | 相同 |
| 统计功能 | O(n) | O(n) | 相同 |

为了统一空表与非空表的操作、简化链表操作的实现，本项目在构建链表的时候在链表最前端加入了头节点head，该节点不存储任何考生信息，仅用于访问链表和提升代码的重用性。

## 2.2 类设计

本项目共定义了两个类，分别是链表节点类linkNode与链表类LinkList。为了提高代码的重用性，程序在定义类的时候使用了类模板以适应不同的数据类型。同时为了方便储存考生的各项信息，程序定义了一个结构体student，该结构体的各项属性即为考生的各项信息，最后对类模板实例化时将数据类型赋为student\*即可。student结构体及两种类模板的示意图如下：



### 2.2.1 链表节点类linkNode

链表节点类的protected属性包含了链表节点的数据域与指针域，其数据域为指向student结构体的指针，通过student结构体的number、name、sex、age、type属性来读出或写入考生的学号、姓名、性别、年龄、报考类型等信息。其指针域为指向下一个linkNode<Type>元素的指针，从而构建出一条元素为linkNode<Type>类型的链表。链表节点类的public属性主要用于申明友元和存放操作。其具有的成员函数如下：

linkNode();    //构造函数

**void** copyNode(**const** linkNode<Type>\* link);    //复制节点信息

**void** inputNode();    //输入节点信息

其中最为核心的是复制节点信息和输入节点信息功能。在输入的过程中需要对用户的输入进行错误处理，一是判断用户输入是否合法，比如对学号之类的int型数据不能输入字符串；二是判断用户的输入是否符合现实逻辑，比如年龄不可能为负数、性别只有两种可能。当系统检测到用户存在错误，程序会给出提示并要求用户重新输入。

### 2.2.2 链表类linkList

链表类linkList的protected属性包含链表的头指针与链表长度。其中头指针用于访问链表，链表长度属性可以在按位插入等操作中快速进行可行性判断，程序通过开辟一个int型的变量length，在链表初始化和增删元素等操作中对length直接处理，将获取链表长度操作的时间复杂度由O(n)降到了O(1)，极好地体现了编程中空间换时间的思想。linkList类的public属性主要存放必要的操作，除了直观呈现的增、删、改、查、统计等操作外，还包含构造与析构函数、打印链表节点信息、循环选择等函数。链表类定义的源代码如下：

//链表类

**template** <**class** Type> **class** linkList {

**protected**:

    linkNode<Type>\* head;    //链表头指针

**int** length = 0;    //链表长度

**public**:

    linkList(linkNode<Type>\* link);    //带一个参数的构造函数

    linkList(linkNode<Type>\* link, **int** number);    //带两个参数的构造函数

    ~linkList();    //析构函数

**void** print();    //打印链表节点信息

    linkNode<Type>\* search(**int** locate);    //按位置查找节点信息

    linkNode<Type>\* searchNumber(**int** number);    //按考号查找节点信息

**void** searchInput();    //查找操作

**void** insert();    //插入操作

**void** cancel();    //删除操作

**void** alter();    //修改操作

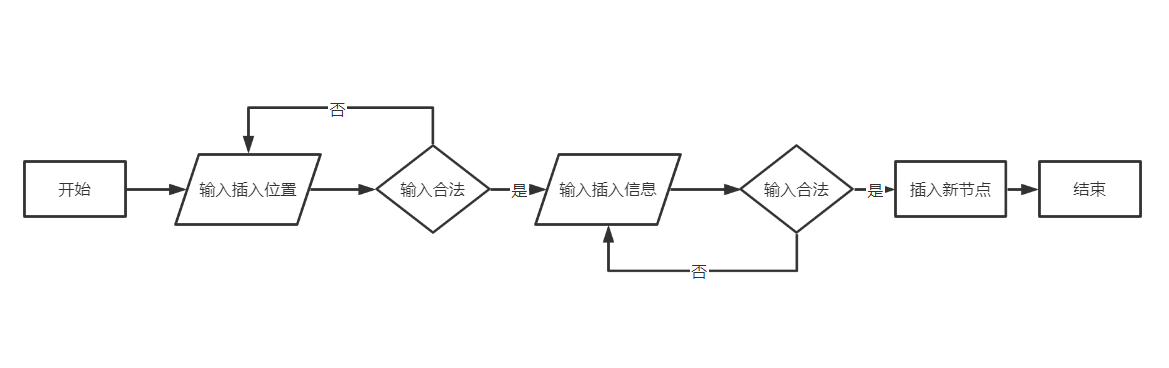
**void** statistics();    //统计操作

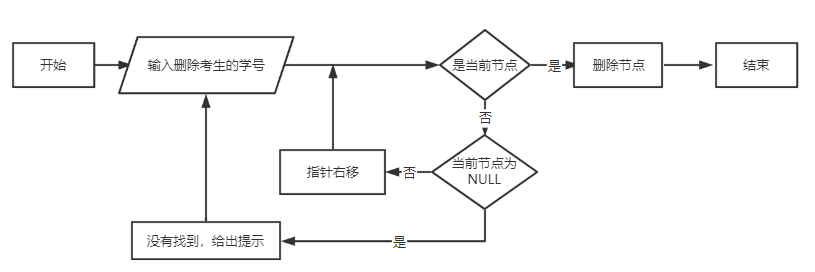
**void** operate();    //循环选择操作

};

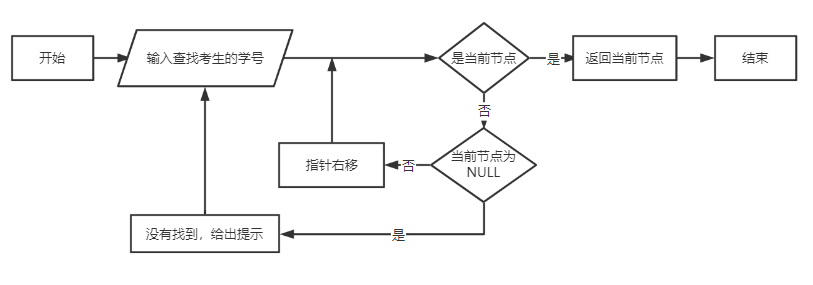
链表类linkList的核心功能实现较为复杂，但都是基于对链表节点的操作实现的，比如增加删除链表节点，修改某一节点的值等。每一项操作都应对用户的输入进行处理，避免因为输入不当导致程序出现问题。以下是链表类核心成员函数实现的功能流程图。

1.插入功能流程图：

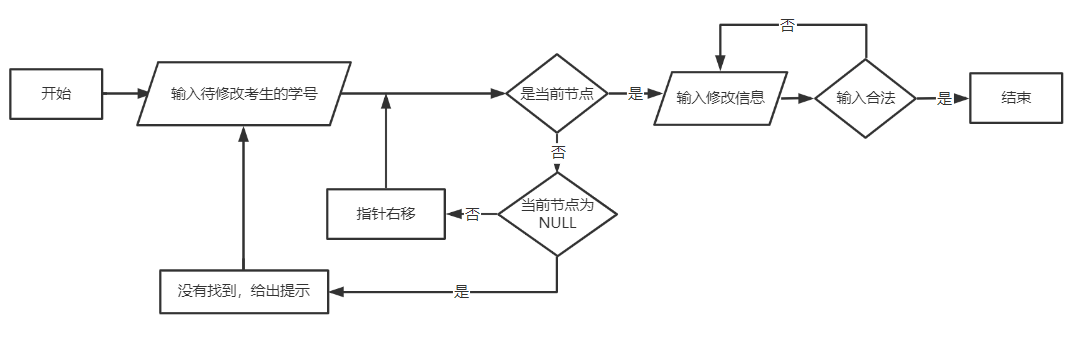
2.删除功能流程图：



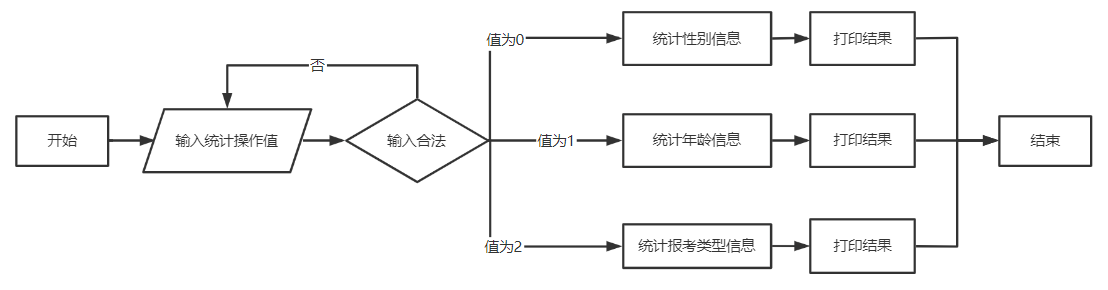
3.查找功能流程图：



4.修改功能流程图：



5.统计功能流程图：



## 2.3 系统设计

本项目是对考试报名管理的简单模拟，通过控制台选项的选择方式对输入考生信息、输出考生信息、查询考生信息、添加考生信息、修改考生信息、删除考生信息等功能的调用。其实现也较为简单，通过input函数输出对该项目功能的整体介绍，帮助用户了解如何正确使用考生报名系统。系统在初始化创建链表类之后调用链表类的循环操作函数，用户输入一个操作数，当操作数为0的时候程序退出，其他情况下分别调用对应的成员函数以实现不同的操作。其中对用户输入的操作数进行错误处理，以提升程序的鲁棒性。

# 3 实现

## 3.1 主体功能实现的预处理

在调用linkList类实现增、删、改、查、统计等主体功能前需先实现一些预先处理操作，其中包括输入节点信息、复制节点信息等功能。由于在主体功能中会频繁对节点进行赋值与复制，故将以上操作封装为linkNode类的成员函数，从而减少程序的代码量，提升程序的可读性。

//复制节点信息

**template** <**class** Type>

**void** linkNode<Type>::copyNode(**const** linkNode<Type>\* link) {

value = **new** student;

value->number = (link->value)->number;

value->name = (link->value)->name;

value->sex = (link->value)->sex;

value->age = (link->value)->age;

value->type = (link->value)->type;

next = link->next;

}

//输入节点信息及错误处理

**template** <**class** Type>

**void** linkNode<Type>::inputNode() {

**while** (1) {

        cin >> value->number;

        cin >> value->name;

        cin >> value->sex;

        cin >> value->age;

        cin >> value->type;

**if** (cin.fail()) {

            cout << "输入错误，请重新输入：";

            cin.clear();

**char** t;

**while** ((t = cin.get()) != '\n');

        }

**else** **if** (value->name.length() >= 8) {

            cout << "姓名过长，请重新输入：";

            cin.clear();

**char** t;

**while** ((t = cin.get()) != '\n');

        }

**else** **if** (value->type.length() >= 20) {

            cout << "报考类型过长，请重新输入：";

            cin.clear();

**char** t;

**while** ((t = cin.get()) != '\n');

        }

**else** **if** (value->age < 0 || value->age >= 120) {

            cout << "年龄输入错误，应该在0-120之间，请重新输入：";

            cin.clear();

**char** t;

**while** ((t = cin.get()) != '\n');

        }

**else** **if** (value->sex != "男" && value->sex != "女") {

            cout << "性别输入错误，请重新输入：";

            cin.clear();

**char** t;

**while** ((t = cin.get()) != '\n');

        }

**else**

**break**;

    }

}

## 3.2 插入功能的实现

### 3.2.1 插入功能实现思路

插入考生信息功能在程序中体现为将新节点插入链表，先通过linkNode类中的inputNode函数完成对新节点的创建，再将该节点插入指定位置。为了统一空表与非空表的操作、简化链表操作的实现，本项目在构建链表的时候在链表最前端加入了不存储任何考生信息的头节点head，在插入过程中只需搜索到该位置的前一个节点，将前一个节点的next指针指向新节点，再将新节点的next指针指向原先节点的后一个节点。当插入位置为链表末尾时，需要将新节点的next指针指向NULL，以表示链表到该节点结束。

### 3.2.2 插入功能实现代码

//插入操作

**template** <**class** Type>

**void** linkList<Type>::insert() {

**int** locate;

    cout << "请输入您要插入的考生位置：";

**while** (1) {

        cin >> locate;

**if** (cin.fail() || locate <= 0 || locate > length + 1) {

            cout << "输入错误，请重新输入：";

            cin.clear();

**char** t;

**while** ((t = cin.get()) != '\n');

        }

**else**

**break**;

    }

    cout << "请依次输入要插入的考生的考号、姓名、性别、年龄及报考类别！" << endl;

    linkNode<Type>\* temp = **new** linkNode<Type>;

**if** (temp == NULL) {

        cout << "初始化失败" << endl;

        exit(-1);

    }

    temp->inputNode();

**if** (locate == 1) {

        length += 1;

        temp->next = head->next;

        head->next = temp;

    }

**else** {

        linkNode<Type>\* node = search(locate);

        temp->next = node->next;

        node->next = temp;

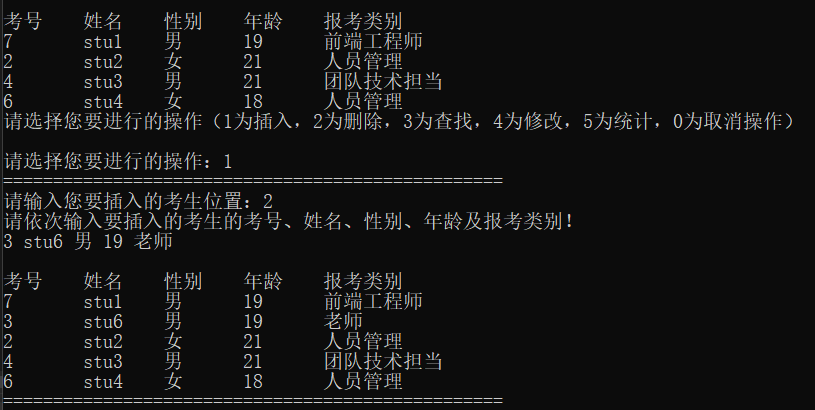
        length += 1;

    }

    print();

}

### 3.2.2 插入功能实现样例



## 3.3 删除功能的实现

### 3.3.1 删除功能实现思路

删除考生信息在程序中体现为删除链表节点，用户在前端输入待删除的考生位置，程序通过搜索函数查找到该位置的前一个节点，将前一个节点的next指针指向该节点

next指针指向节点，并将该指针的存储空间通过delete命令释放。若待删除的节点为链表的最后一个节点，则将其前一个节点的next指针指向NULL。

### 3.3.2 删除功能实现代码

//删除操作

**template** <**class** Type>

**void** linkList<Type>::cancel() {

**if** (length == 0) {

cout << "无任何可删除数据" << endl;

return;

}

**int** locate;

    cout << "请输入要删除的考生的考号：";

**while** (1) {

        cin >> locate;

**if** (cin.fail()) {

            cout << "输入错误，请重新输入：";

            cin.clear();

**char** t;

**while** ((t = cin.get()) != '\n');

        }

**else** **if** (!searchNumber(locate) && locate != (head->value)->number) {

            cout << "搜索不到该考生号，请重新输入：";

            cin.clear();

**char** t;

**while** ((t = cin.get()) != '\n');

        }

**else**

**break**;

    }

    linkNode<Type>\* node = searchNumber(locate)->next;

cout << "你删除的考生信息是：" << locate <<

" " << (node->value)->name << " " << (node->value)->sex << " " << (node->value)->age << " " << (node->value)->type << endl;

**if** (((searchNumber(locate)->next)->next))

        searchNumber(locate)->next = (searchNumber(locate)->next)->next;

**else**

        searchNumber(locate)->next = NULL;

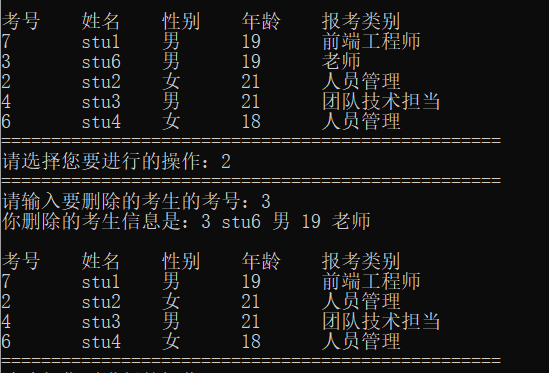
**delete** node;

    length -= 1;

    print();

}

### 3.2.2 删除功能实现样例



## 3.4 修改功能的实现

### 3.4.1 修改功能实现思路

修改考生信息在程序中体现为修改链表节点的数据域。用户输入待修改的考生考号，程序遍历链表，如若未查找到链表中存在某一节点的考号与用户输入相同，则给出提示该考生不存在并让用户重新输入。如若查找到则将用户输入的信息赋值给该节点数据域，即修改其student\* value所指向结构体变量的各属性值。

### 3.4.2 修改功能实现代码

//修改操作

**template** <**class** Type>

**void** linkList<Type>::alter() {

**int** locate;

    cout << "请输入要修改的考生的考号：";

**while** (1) {

        cin >> locate;

**if** (cin.fail()) {

            cout << "输入错误，请重新输入：";

            cin.clear();

**char** t;

**while** ((t = cin.get()) != '\n');

        }

**else** **if** (!searchNumber(locate) && locate != (head->value)->number) {

            cout << "搜索不到该考生号，请重新输入：";

            cin.clear();

**char** t;

**while** ((t = cin.get()) != '\n');

        }

**else**

**break**;

    }

    cout << "请依次输入修改后该考生的考号、姓名、性别、年龄及报考类别！" << endl;

**if** (locate == (head->value)->number) {

        head->inputNode();

    }

**else** {

        linkNode<Type>\* temp = searchNumber(locate)->next;

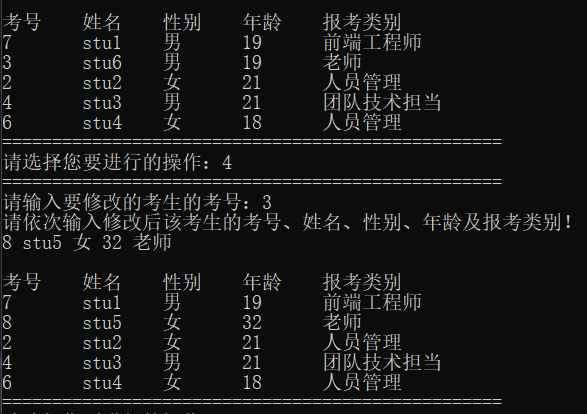
        temp->inputNode();

    }

    print();

}

### 3.2.2 修改功能实现样例



## 3.5 查找功能的实现

### 3.5.1 查找功能实现思路

查找考生信息在程序中体现为遍历链表并返回和用户输入考号相同的节点。如若未查找到链表中存在某一节点的考号与用户输入相同，则给出提示该考生不存在并让用户重新输入。如若存在则将该节点的数据域打印出来。

### 3.5.2 查找功能实现代码

//查找操作

**template** <**class** Type>

**void** linkList<Type>::searchInput() {

**int** locate;

    cout << "请输入要查找的考生考号：";

**while** (1) {

        cin >> locate;

**if** (cin.fail()) {

            cout << "输入错误，请重新输入：";

            cin.clear();

**char** t;

**while** ((t = cin.get()) != '\n');

        }

**else** **if** (!searchNumber(locate) && locate != (head->value)->number) {

            cout << "搜索不到该考生号，请重新输入：";

            cin.clear();

**char** t;

**while** ((t = cin.get()) != '\n');

        }

**else**

**break**;

    }

**if** (locate == (head->value)->number) {

        cout << "考号    姓名    性别    年龄    报考类别" << endl;

        cout << (head->value)->number << " " << (head->value)->name << " " << (head->value)->sex << " " << (head->value)->age << " " << (head->value)->type << endl;

    }

**else** {

        cout << "考号    姓名    性别    年龄    报考类别" << endl;

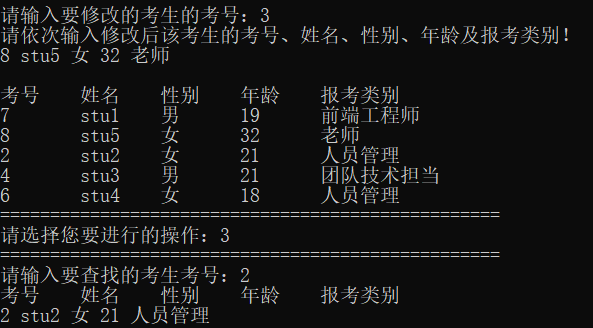
        linkNode<Type>\* node = searchNumber(locate)->next;

        cout << (node->value)->number << " " << (node->value)->name << " " << (node->value)->sex << " " << (node->value)->age << " " << (node->value)->type << endl;

    }

}

### 3.5.3 查找功能实现样例



## 3.6 统计功能的实现

### 3.6.1 统计功能实现思路

统计功能在程序中体现为遍历搜索链表，将具有相同属性的链表节点构建成新的链表，并将这些链表一一打印。本项目统计功能具有三项子功能，分别是统计考生性别信息、统计考生年龄信息、统计考生报考类型信息，这三项子功能的具体实现有所差别。在统计性别信息的实现中，由于性别只有两种分类，故可预先建立两个链表类对象man、woman，在遍历考生信息时只需对性别做出判断并将其拼接至对应的链表类对象即可。而年龄信息相对性别可能性更多，为了提高搜索效率，可建立哈希表，年龄为i的元素存储在数组的第i项上，输出时将不为空的数组元素输出即可。统计考生报考信息可建立一个链表指针数组存放不同报考类型对应的链表，在遍历考生信息过程中若第一次读取到某报考类型则将其存放在链表指针数组的最后一个不为空的位置上。

### 3.6.2 统计功能实现代码

**template** <**class** Type>

**void** linkList<Type>::statistics() {

**int** type;

    cout << "请输入需要统计的数据（0为性别，1为年龄，2为报考类别）：";

**while** (1) {

        cin >> type;

**if** (cin.fail() || type < 0 || type > 2) {

            cout << "输入错误，请重新输入：";

            cin.clear();

**char** t;

**while** ((t = cin.get()) != '\n');

        }

**else**

**break**;

    }

**if** (type == 0) {    //对男性女性分别创建链表存储信息

**int** step = 0;

**int** male = 0, female = 0;

        linkNode<Type>\* man = **new** linkNode<Type>;

        linkNode<Type>\* woman = **new** linkNode<Type>;

**if** (man == NULL || woman == NULL) {

            cout << "初始化失败" << endl;

            exit(-1);

        }

        linkNode<Type>\* temp = head->next;

        linkNode<Type>\* temp1 = man;

        linkNode<Type>\* temp2 = woman;

**while** (step < length) {

            linkNode<Type>\* node = **new** linkNode<Type>;

            node->copyNode(temp);

**if** (node == NULL) {

                cout << "初始化失败" << endl;

                exit(-1);

            }

**if** ((temp->value)->sex== "男") {

                temp1->next = node;

                temp1 = node;

                male += 1;

            }

**else** {

                temp2->next = node;

                temp2 = node;

                female += 1;

            }

            temp = temp->next;

            step += 1;

        }

        temp1->next = NULL;

        temp2->next = NULL;

        cout << "性别为男：共" << male << "条数据：" << endl;

        linkList<Type> linkMan(man);

        linkMan.print();

        cout << "性别为女：共" << female << "条数据：" << endl;

        linkList<Type> linkWoman(woman);

        linkWoman.print();

    }

**if** (type == 1) {    //建立哈希表将年龄为i的考生信息存储在数组第i个元素上

**int** step = 0;

        linkNode<Type>\* temp = head->next;

        linkNode<Type>\* age[120];

        linkNode<Type>\* ageTemp[120];

**int** ageSum[120] = { 0 };

**for** (**int** i = 0; i < 120; i++) {

            age[i] = NULL;

            ageTemp[i] = NULL;

        }

**while** (step < length) {

**if** (age[(temp->value)->age] == NULL) {

                age[(temp->value)->age] = **new** linkNode<Type>;

**if** (age[(temp->value)->age] == NULL) {

                    cout << "初始化失败" << endl;

                    exit(-1);

                }

                ageTemp[(temp->value)->age] = age[(temp->value)->age];

            }

            linkNode<Type>\* node = **new** linkNode<Type>;

            node->copyNode(temp);

**if** (node == NULL) {

                cout << "初始化失败" << endl;

                exit(-1);

            }

            ageTemp[(temp->value)->age]->next = node;

            ageTemp[(temp->value)->age] = node;

            ageSum[(temp->value)->age] += 1;

            temp = temp->next;

            step += 1;

        }

**for** (**int** i = 0; i < 120; i++) {

**if** (ageSum[i] != 0) {

**if** (ageTemp[i] != NULL)

                    ageTemp[i]->next = NULL;

                cout << "年龄为" << i << "：共" << ageSum[i] << "条数据：" << endl;

                linkList<Type> link(age[i]);

                link.print();

            }

        }

    }

**if** (type == 2) {    //建立链表数组存放考生报考类别信息

**int** step = 0, key = 0;

        linkNode<Type>\* temp = head->next;

        linkNode<Type>\* work[100];

        linkNode<Type>\* workTemp[100];

**int** workSum[100] = { 0 };

        string workType[100];

**for** (**int** i = 0; i < 100; i++) {

            work[i] = NULL;

            workTemp[i] = NULL;

        }

**while** (step < length) {

**int** flag = 0;

**int** keyNow = 0;

**for** (**int** i = 0; i < key; i++) {

**if** ((temp->value)->type == workType[i]) {

                    keyNow = i;

                    flag = 1;

                }

            }

**if** (flag == 0) {

                workType[key] = (temp->value)->type;

                work[key] = **new** linkNode<Type>;

**if** (work[key] == NULL) {

                    cout << "初始化失败" << endl;

                    exit(-1);

                }

                workTemp[key] = work[key];

                keyNow = key;

                key += 1;

            }

            linkNode<Type>\* node = **new** linkNode<Type>;

            node->copyNode(temp);

**if** (node == NULL) {

                cout << "初始化失败" << endl;

                exit(-1);

            }

            workTemp[keyNow]->next = node;

            workTemp[keyNow] = node;

            workSum[keyNow] += 1;

            temp = temp->next;

            step += 1;

        }

**for** (**int** i = 0; i < 100; i++) {

**if** (workSum[i] != 0) {

**if** (workTemp[i] != NULL)

                    workTemp[i]->next = NULL;

                cout << "报考类型为" << workType[i] << "：共" << workSum[i] << "条数据：" << endl;

                linkList<Type> link(work[i]);

                link.print();

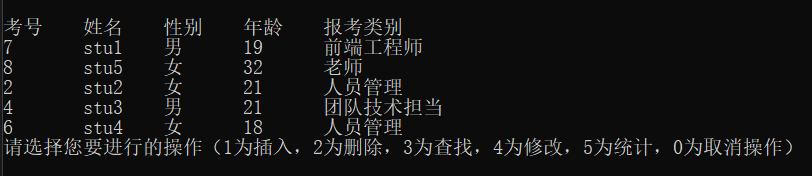
            }

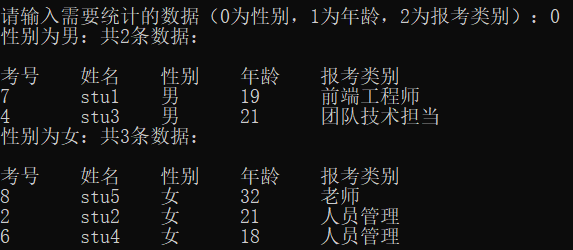
        }

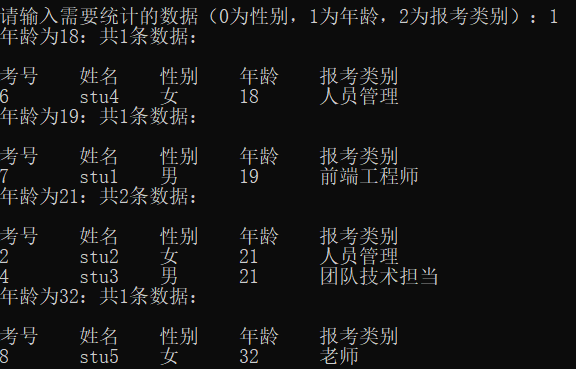
}

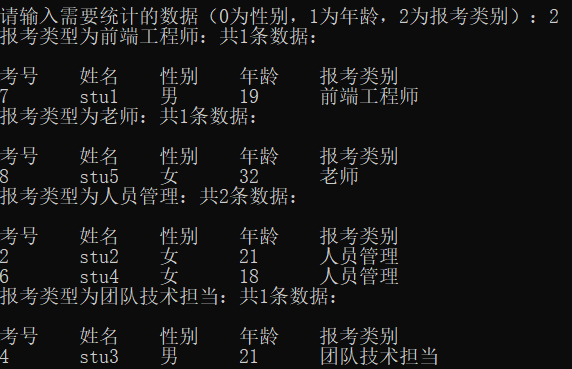
}

### 3.6.2 统计功能实现样例









# 4 测试

## 4.1 初始化测试

### 4.1.1 正常输入情况

测试案例：

系统初始化时建立一个考生数量为4的链表。输入信息：

7 stu1 男 19 前端工程师

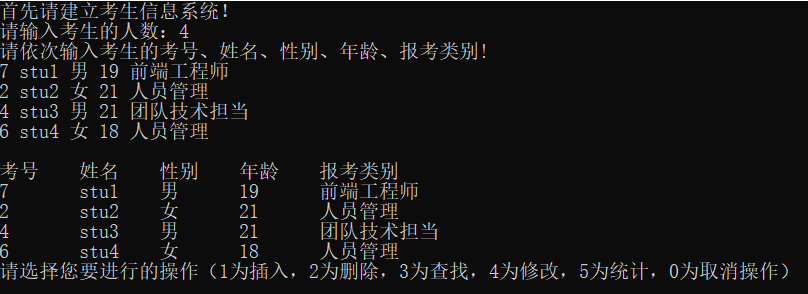
2 stu2 女 21 人员管理

4 stu3 男 21 团队技术担当

6 stu4 女 18 人员管理

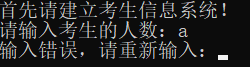
预期系统将输入的数据格式化输出。

测试结果：

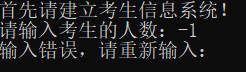


### 4.1.2 错误输入情况

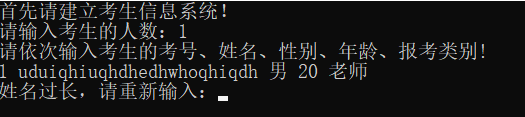
输入考生人数不为数字：



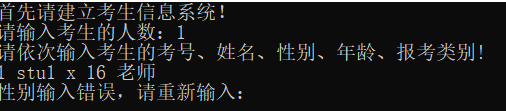
输入考生人数为负数：



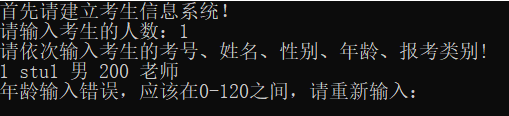
输入考生姓名过长：



输入考生性别不为‘男’或‘女’：



输入考生年龄不符合实际情况：



## 4.2 插入功能测试

系统初始数据：

7 stu1 男 19 前端工程师

2 stu2 女 21 人员管理

4 stu3 男 21 团队技术担当

6 stu4 女 18 人员管理

### 4.2.1 普通插入

测试案例：

在位置为2处插入考生信息：

8 stu5 女 32 老师

预期结果：

7 stu1 男 19 前端工程师

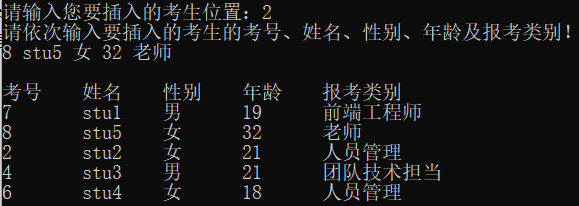
8 stu5 女 32 老师

2 stu2 女 21 人员管理

4 stu3 男 21 团队技术担当

6 stu4 女 18 人员管理

测试结果：



### 4.2.2 边界测试（在链表头部插入）

测试案例：

在位置为0处插入考生信息：

8 stu5 女 32 老师

预期结果：

8 stu5 女 32 老师

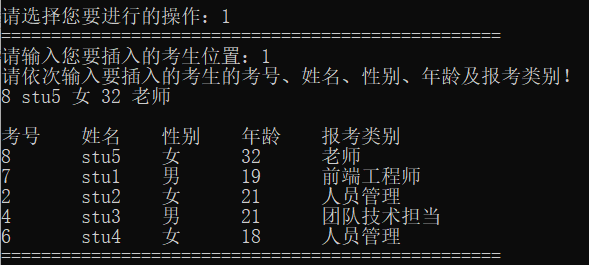
7 stu1 男 19 前端工程师

2 stu2 女 21 人员管理

4 stu3 男 21 团队技术担当

6 stu4 女 18 人员管理

测试结果：



### 4.2.3 边界测试（在链表尾部插入）

测试案例：

在位置为5处插入考生信息：

8 stu5 女 32 老师

预期结果：

7 stu1 男 19 前端工程师

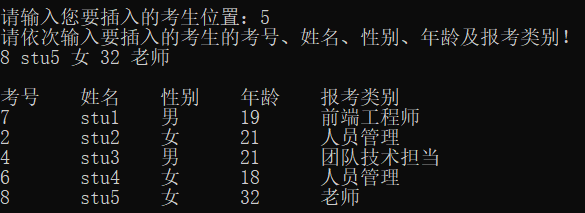
2 stu2 女 21 人员管理

4 stu3 男 21 团队技术担当

6 stu4 女 18 人员管理

8 stu5 女 32 老师

测试结果：



### 4.2.4 错误测试（插入位置不存在）

测试案例：

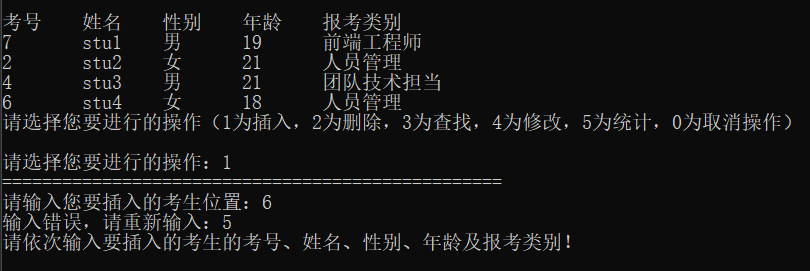
在位置为6处插入考生信息：

8 stu5 女 32 老师

预期结果：

报错并提示用户重新输入。

测试结果：



## 4.3 删除功能测试

系统初始数据：

7 stu1 男 19 前端工程师

2 stu2 女 21 人员管理

4 stu3 男 21 团队技术担当

6 stu4 女 18 人员管理

### 4.3.1 普通删除

测试案例：

删除考号为2的考生信息。

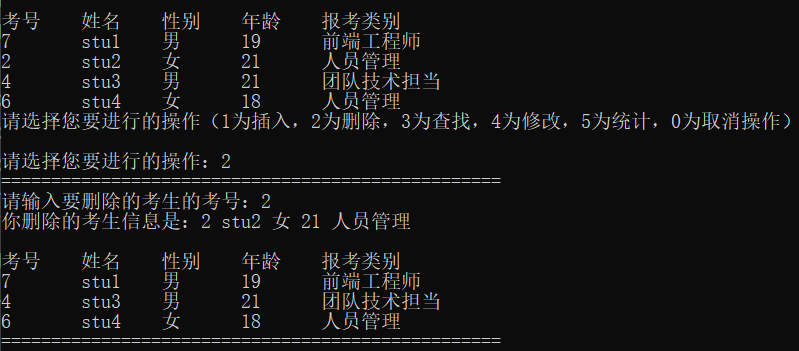
预期结果：

7 stu1 男 19 前端工程师

4 stu3 男 21 团队技术担当

6 stu4 女 18 人员管理

测试结果：



### 4.3.2 边界测试（删除链表头部节点）

测试案例：

删除考号为7的考生信息。

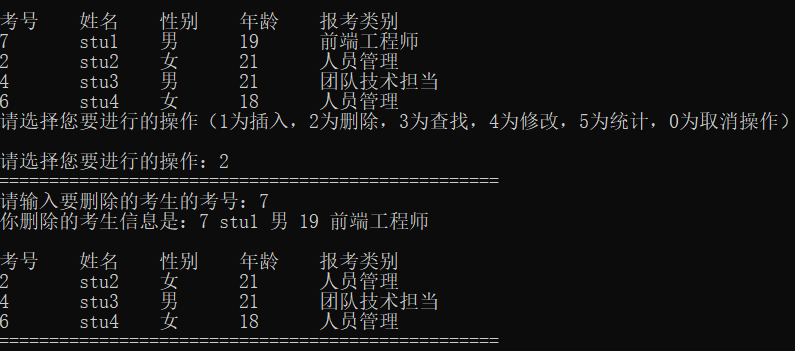
预期结果：

2 stu2 女 21 人员管理

4 stu3 男 21 团队技术担当

6 stu4 女 18 人员管理

测试结果：



### 4.3.3 边界测试（删除链表尾部节点）

测试案例：

删除考号为6的考生信息。

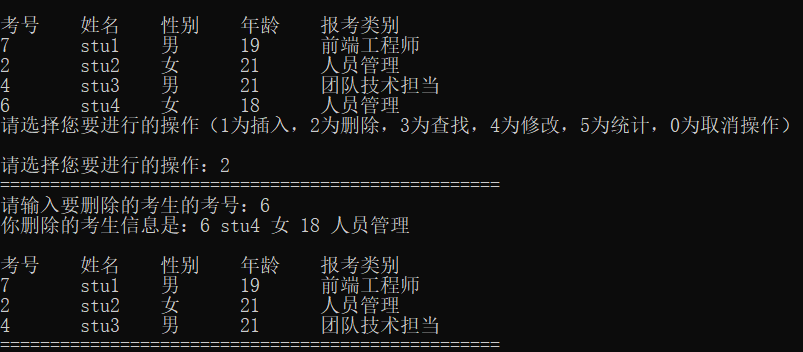
预期结果：

7 stu1 男 19 前端工程师

2 stu2 女 21 人员管理

4 stu3 男 21 团队技术担当

测试结果：



### 4.3.4 边界测试（删除空链表）

预先将链表删除至只剩头节点。

测试案例：

删除考号为6的考生信息。

预期结果：

提示无任何可删除数据并跳出操作。

测试结果：

### 

### 4.3.5 错误测试（删除考生考号不存在）

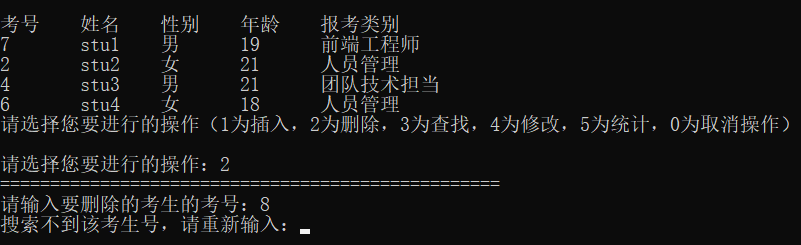
测试案例：

删除考号为8的考生信息：

预期结果：

报错并提示用户重新输入。

测试结果：



## 4.4 修改功能测试

系统初始数据：

7 stu1 男 19 前端工程师

2 stu2 女 21 人员管理

4 stu3 男 21 团队技术担当

6 stu4 女 18 人员管理

### 4.4.1 普通修改

测试案例：

修改考号为2的考生信息为：

8 stu5 女 32 老师

预期结果：

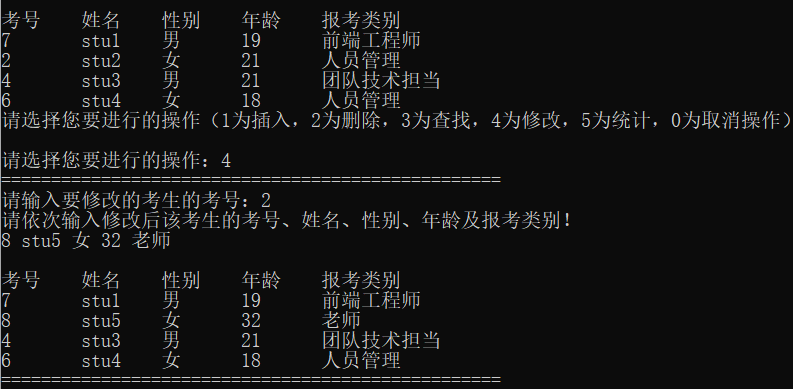
7 stu1 男 19 前端工程师

8 stu5 女 32 老师

4 stu3 男 21 团队技术担当

6 stu4 女 18 人员管理

测试结果：



### 4.4.2 边界测试（修改链表头部节点）

测试案例：

修改考号为7的考生信息为：

8 stu5 女 32 老师

预期结果：

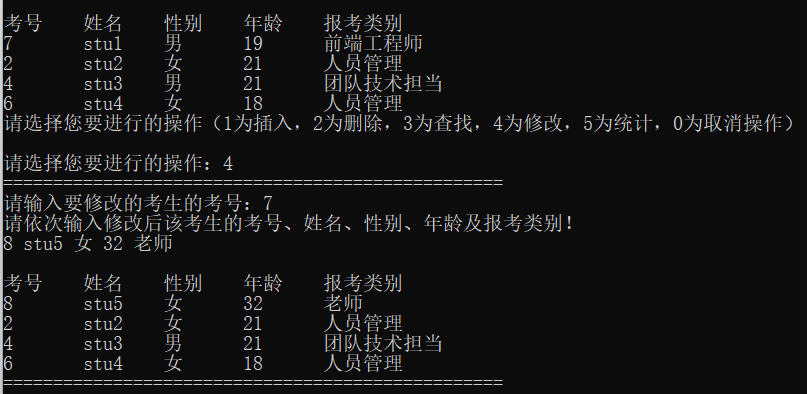
8 stu5 女 32 老师

2 stu2 女 21 人员管理

4 stu3 男 21 团队技术担当

6 stu4 女 18 人员管理

测试结果：



### 4.4.3 边界测试（修改链表尾部节点）

测试案例：

修改考号为6的考生信息为：

8 stu5 女 32 老师

预期结果：

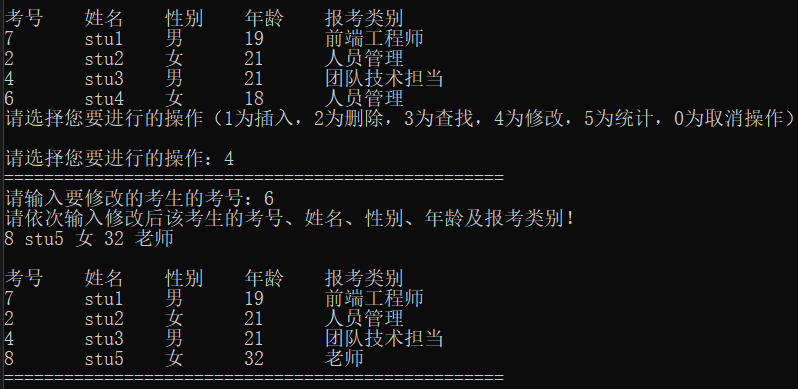
7 stu1 男 19 前端工程师

2 stu2 女 21 人员管理

4 stu3 男 21 团队技术担当

8 stu5 女 32 老师

测试结果：



### 4.4.4 错误测试（修改考生考号不存在）

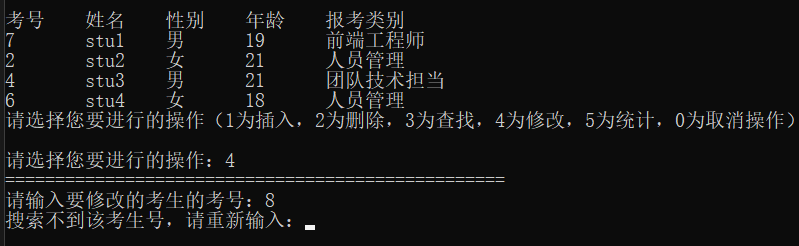
测试案例：

修改考号为8的考生信息。

预期结果：

报错并提示用户重新输入。

测试结果：



## 4.5 查找功能测试

系统初始数据：

7 stu1 男 19 前端工程师

2 stu2 女 21 人员管理

4 stu3 男 21 团队技术担当

6 stu4 女 18 人员管理

### 4.5.1 普通查找

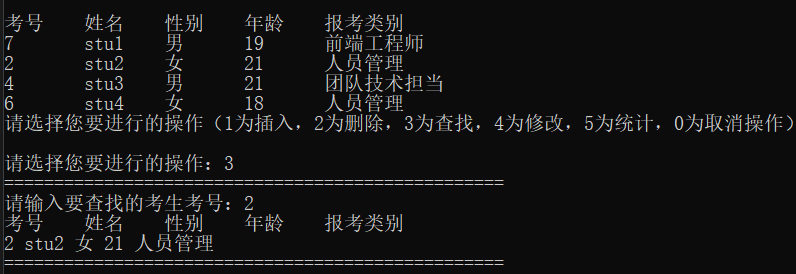
测试案例：

查找考号为2的考生信息。

预期结果：

2 stu2 女 21 人员管理

测试结果：



### 4.5.2 边界测试（查找链表头部节点）

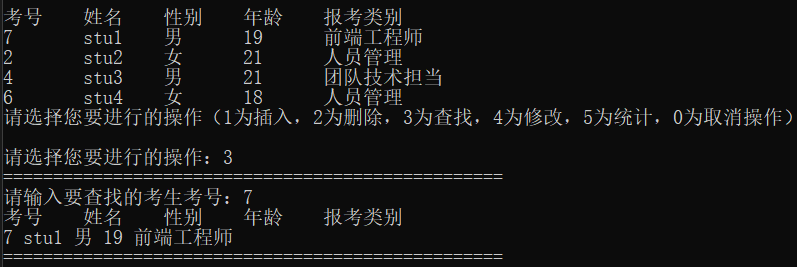
测试案例：

查找考号为7的考生信息。

预期结果：

7 stu1 男 19 前端工程师

测试结果：



### 4.5.3 边界测试（查找链表尾部节点）

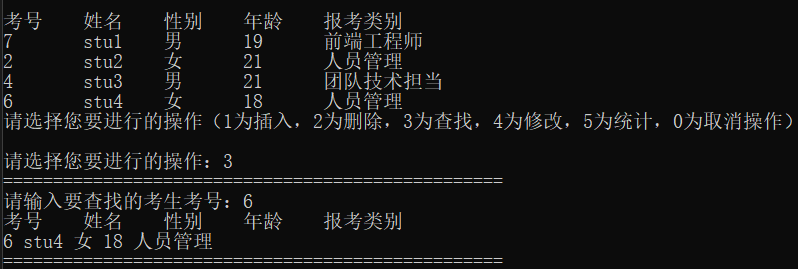
测试案例：

查找考号为6的考生信息。

预期结果：

6 stu4 女 18 人员管理

测试结果：



### 4.5.4 错误测试（查找考生考号不存在）

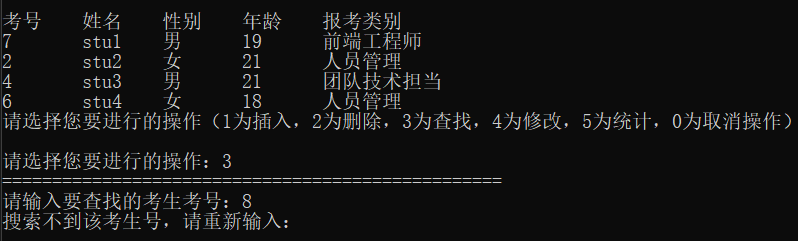
测试案例：

查找考号为8的考生信息。

预期结果：

报错并提示用户重新输入。

测试结果：



## 4.6 统计功能测试

系统初始数据：

7 stu1 男 19 前端工程师

8 stu5 女 32 老师

2 stu2 女 21 人员管理

4 stu3 男 21 团队技术担当

6 stu4 女 18 人员管理

测试结果：

