**数据结构课程设计**

**项目说明文档**

**考生报名系统**

作 者 姓 名： 苏家铭

学 号： 2151299

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 项目分析 1](#_Toc14343)

[1.1 项目背景 1](#_Toc18466)

[1.2项目要求 1](#_Toc27217)

[1.2.1 功能要求 1](#_Toc25903)

[1.2.2 项目实例 1](#_Toc1412)

[2 项目设计及实现 2](#_Toc4651)

[2.1 数据结构设计思路 2](#_Toc494)

[2.2 类设计 2](#_Toc5486)

[2.2.1 链表 2](#_Toc1737)

[2.2.2 链式栈 4](#_Toc29337)

[2.2.3 AVL树 5](#_Toc25908)

[2.3 项目算法 10](#_Toc18992)

[2.3.1 实现思路 10](#_Toc2049)

[2.3.2 代码实现 11](#_Toc24245)

[3 项目测试 13](#_Toc26323)

[3.1 初始化考生信息 13](#_Toc13244)

[3.2 插入考生信息 14](#_Toc16233)

[3.3 删除考生信息 15](#_Toc7835)

[3.4 查找考生信息 15](#_Toc11785)

[3.5 修改考生信息 16](#_Toc7585)

[3.6 统计考生数量并展示 16](#_Toc18017)

[3.7 退出系统 17](#_Toc12792)

[3.8 边界测试1：初始化考生人数为负数 17](#_Toc6237)

[3.9 边界测试2：输入考生信息错误（年龄或学号为负数） 18](#_Toc6557)

[3.10 边界测试3：未查找到考生信息 18](#_Toc10435)

[3.11 边界测试4：删除不存在的考生信息 18](#_Toc29203)

[3.12 边界测试5：修改不存在的考生信息 19](#_Toc25488)

[3.13 边界测试6：修改后的考生信息有误 19](#_Toc20926)

[3.14 边界测试7：选择操作非法 20](#_Toc32371)

[4 算法性能分析 20](#_Toc5386)

[4.1 正确性 20](#_Toc13265)

[4.2 可使用性 21](#_Toc17863)

[4.3 可读性 21](#_Toc17207)

[4.4 效率 21](#_Toc30302)

[4.5 健壮性 21](#_Toc30677)

# 1 项目分析

## 项目背景

考试报名工作给各高校报名工作带来了新的挑战，给教务管理部门增加了很大的工作量。在我们日常生活中考试无处不在，并且每次考试报名工作都是一个大工程，如果能借助计算机使得繁杂的考生信息得以稳定快速的存储，那么很多工作部门的工作效率将会得到提高，从而实现此项目具有巨大的社会效益。

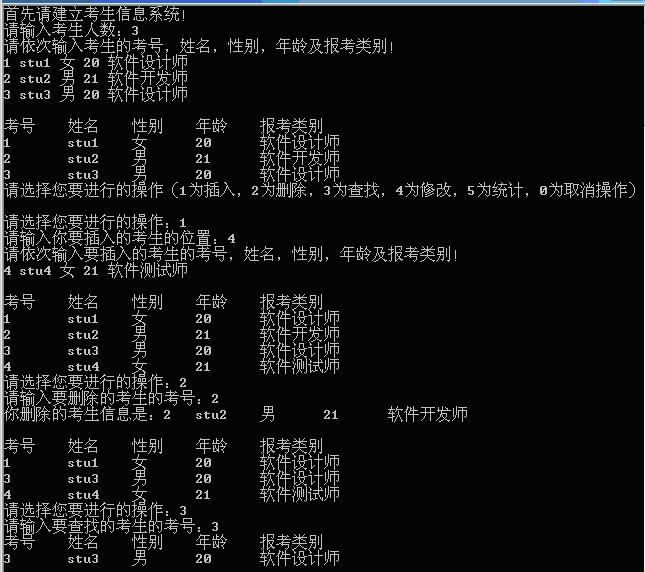
本项目是对考试报名管理的简单模拟，用控制台选项的选择方式完成下列功能：输入考生信息；输出考生信息；查询考生信息；添加考生信息；修改考生信息；删除考生信息。

## 1.2项目要求

### 1.2.1 功能要求

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。项目在设计时应首先确定系统的数据结构，定义类的成员变量和成员函数；然后实现各成员函数以完成对数据操作的相应功能；最后完成主函数以验证各个成员函数的功能并得到运行结果。（建议采用链表实现）

### 1.2.2 项目实例



# 2 **项目设计及实现**

## 2.1 **数据结构设计思路**

考生报名系统是一个繁杂而多功能的系统，在存储这个系统时可采用链表、树、图、散列等数据结构，但其中查询、添加、修改、删除考生信息都需要搜索整个考生信息群，如此频繁的操作不由得我们想到使用AVL树来存储，已实现快速地搜索考生信息，完成各类操作。

而在实现AVL数的插入和删除操作时需要用到栈（stack）来记录查找路径，

在栈中又使用了链表（list）来实现链式栈的物理结构。

## 2.2 **类设计**

### 2.2.1 链表

虽然带头结点的双向循环链表结构复杂，但使用代码后会发现许多优势。该链表一般包括两个抽象数据类型（ADT）：链表结点类（DblNode）与链表类（DblList），而两个类之间的耦合关系可以采用嵌套、复合等多种关系。若结点类定义为class那么需要在类内定义友元函数链表类才能访问，为了实现代码的复用性，本人使用结构体来定义链表的结点，此方法虽然使得DblNode类失去了封装性，但链表类可以直接访问结点中的数据成员。

节点类数据成员：

template<class T>

T data;//链表结点数据

DblNode<T>\* lLink,\*rLink;//链指针域,前驱（左链）、后继（右链）

链表类数据成员：

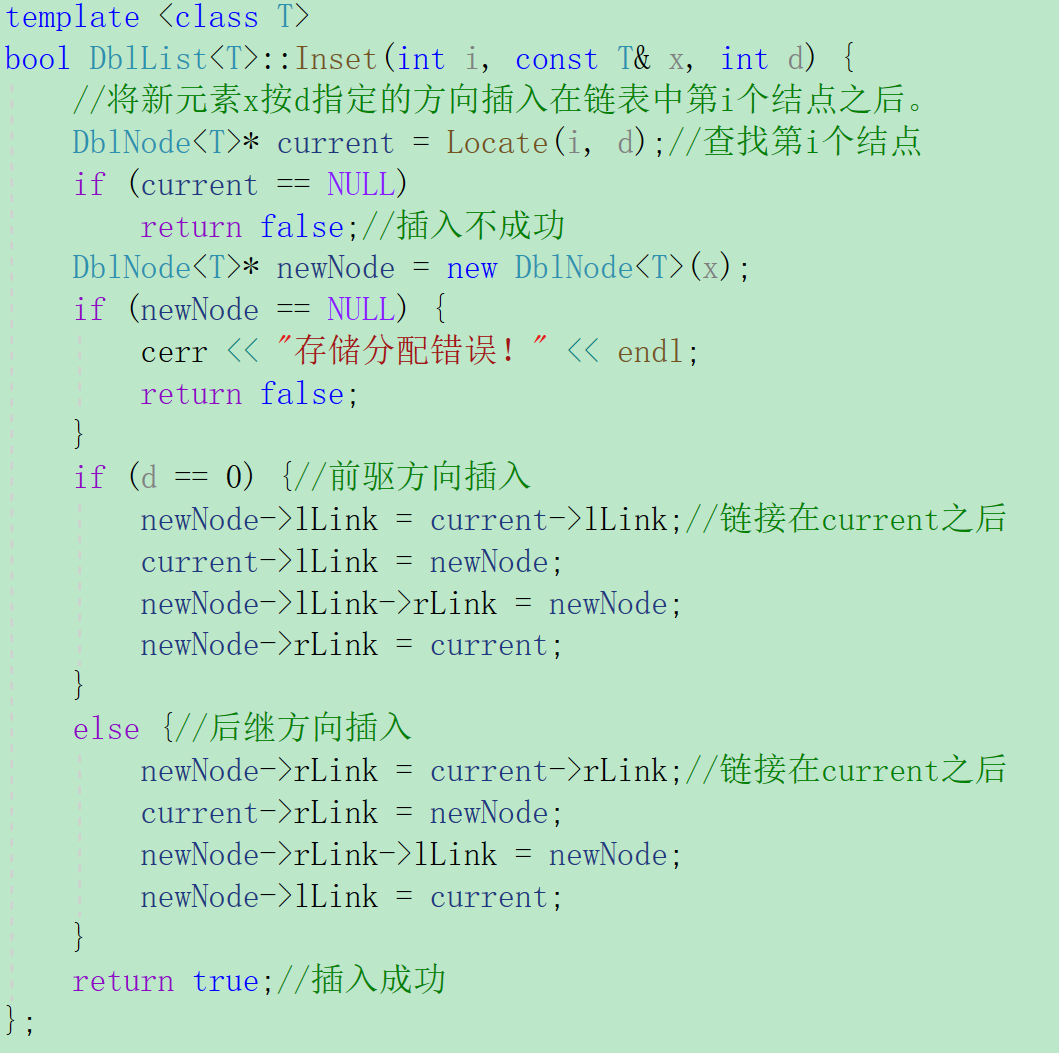
DblNode<T>\* first;//链表头结点

该链表已经实现19种功能：置空、计算长度、返回头结点、设置头结点、搜索、定位、返回元素地址、修改元素的值、插入、删除、置空、判空、判满、尾部插入、头部删除、向后插入i个元素、向后插入若干元素（直到遇到endTag）、输出链表、等号重载。

具体函数为：

* void makeEmpty();//将链表置为空表
* int Length()const;//计算双链表的长度
* void setHead(DblNode<T>\* ptr)//设置附加头结点地址
* DblNode<T>\* getHead()const//返回附加头结点地址
* DblNode<T>\* Search(const T& x);//在链表中沿后继寻找等于数据x的元素
* DblNode<T>\* Locate(int i,int d);//在链表中定位序号为i（>=0）的结点，d=0按前驱方向，d!=0按后继方向
* bool getDate(int i, T& x,int d);//取出按d方向第i个元素的地址
* void setData(int i, T& x,int d);//用x修改按d方向第i个元素的值
* bool Inset(int i, const T& x,int d);//在第i个元素后插入x,d=0按前驱方向，d!=0按后继方向
* bool Remove(int i, T& x,int d);//删除第i个元素,x返回该元素的值,d=0按前驱方向，d!=0按后继方向
* bool IsEmpty()const//判表空否?空则返回true
* bool IsFull()const {return false;}//判表满否？不满则返回false
* void push\_back(T data);//在尾部插入一个数据
* void push\_front(T data);//在前部插入一个数据
* void input\_num(int i);//输入i个元素
* void input\_endTag(T endTag);//输入若干元素，直到遇到endTag
* void output(int d);//输出
* DblList<T>& operator= (const DblList<T>&L);//重载函数：赋值

插入算法的代码实现：



主要算法的时间复杂度：

* 插入：O(n)
* 删除：O(n)
* 搜索：O(n)
* 遍历：O(n)
* 修改：O(n)

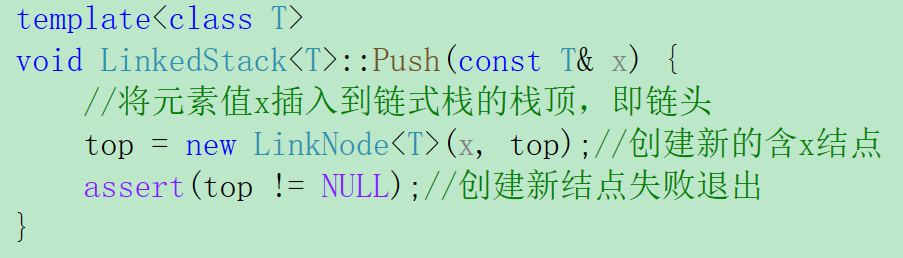
### 2.2.2 链式栈

[栈](https://so.csdn.net/so/search?q=%E6%A0%88%E5%92%8C%E9%98%9F%E5%88%97&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/qq_41575507/article/details/_blank)是重要的线性结构，从数据结构的角度看，栈也是线性表，其特殊性在于栈的基本操作是线性表的子集。这是一种是操作受限的线性表，因此，可称为限定性的数据结构。但从数据类型角度看，他是和[线性](https://so.csdn.net/so/search?q=%E7%BA%BF%E6%80%A7&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/qq_41575507/article/details/_blank)表大不相同的重要的抽象数据类型。栈(stack)是限定仅在表尾进行插入或者删除的线性表。对于栈来说，表尾端称为栈顶(top)，表头端称为栈低(bottom)。因为栈限定在一端进行插入或者删除，所以栈又被称为后进先出的线性表（LIFO）。

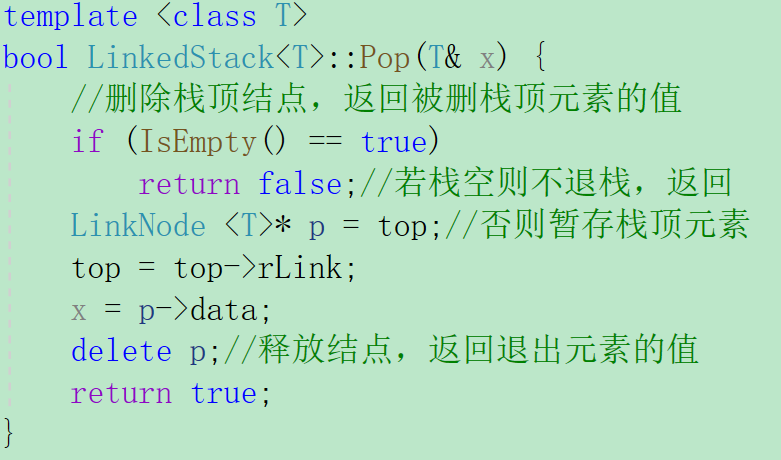
本人基于链表实现了链式栈的进栈、退栈、返回栈顶元素、返回栈内元素个数、置空、判空以及左移运算符的重载操作

* void Push(const T& x);//一个元素进栈
* bool Pop(T& x);//一个元素退栈
* bool getTop(T& x)const;//返回栈顶元素
* bool IsEmpty()const //判空
* int getSize()const;//返回栈的元素个数
* void makeEmpty();//清空栈
* friend ostream& operator << (ostream& os, LinkedStack<T>& s);//输出

进栈算法代码实现：



出栈算法代码实现：



主要算法的时间复杂度：

* 进栈：O(1)
* 退栈：O(1)
* 返回栈顶元素：O(1)
* 返回元素个数：O(n)

### 2.2.3 AVL树

树是一种非线性的数据结构，它是由n（n>=0）个有限结点组成一个具有层次关系的集合。把它叫做树是因为它看起来像一棵倒挂的树，也就是说它是根朝上，而叶朝下的。二叉树是一个结点最多只有两个结点的树，而且二叉树的子树有左右之分，且左右不能颠倒。AVL树又被称为平衡二叉树，是在二叉树的基础上每个结点增设了一个成员balance，用于指示左右子树的高度差值，若|balance|>=2则该子树不平衡，需要通过旋转得到一颗新的平衡二叉树以保持|balance|<2。引入AVL树的目的，是为了提高二叉树搜索树的效率，减少树的平均搜索长度，在AVL树中搜索得到一个元素的时间复杂度为O(log2n),优于在链表中的时间复杂度O(n)，适用于需要反复查找元素的情况。

对于树中的每一个结点，这里定义了一个结点结构体，其数据成员包括：结点数据、结点的左右指针、以及平衡因子。

AVL树结点结构体数据成员：

template<class E, class K>//E是结点，K是结点的关键码

E data;//结点数据

AVLNode<E, K>\* left, \* right;//左右指针

int bf;//平衡因子balance factor

AVL树类数据成员：

AVLNode<E, K>\* root;//根指针

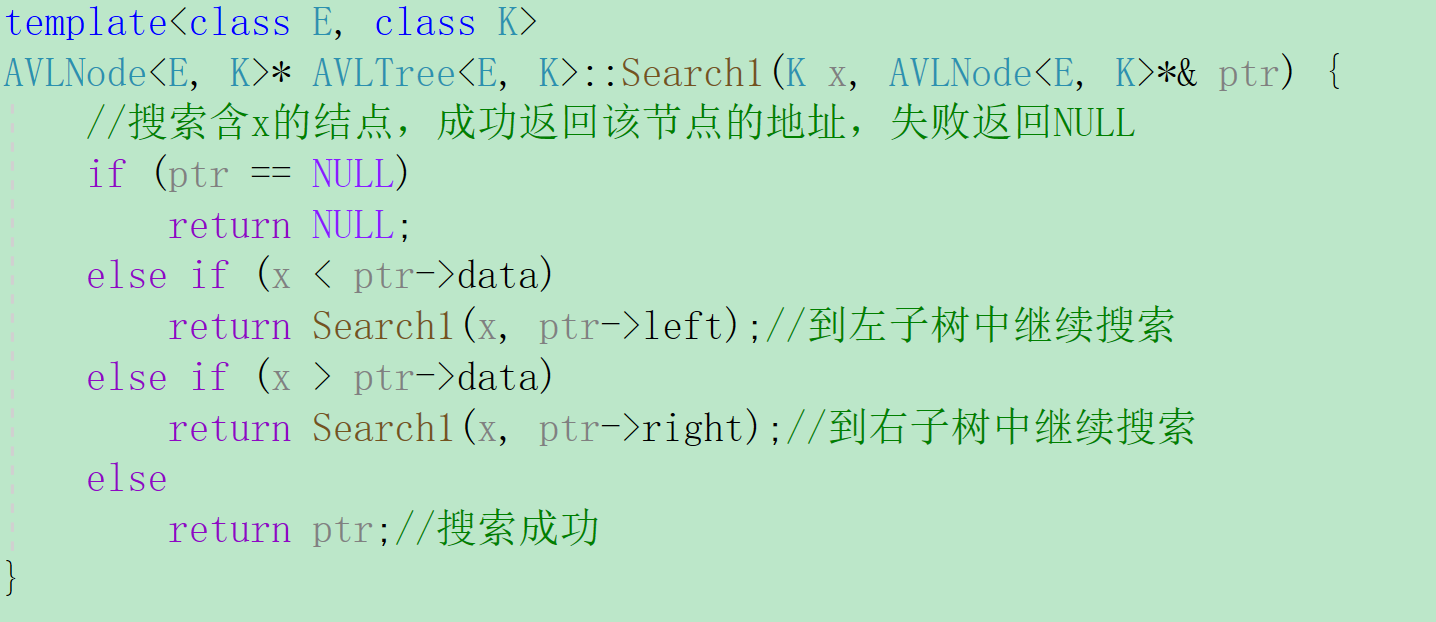
int tree\_num;//结点个数

本AVL已经实现包括插入、删除、（单/双）旋转、查找、计数等18个功能（包括内部私有函数和对外接口）。其函数声明如下：

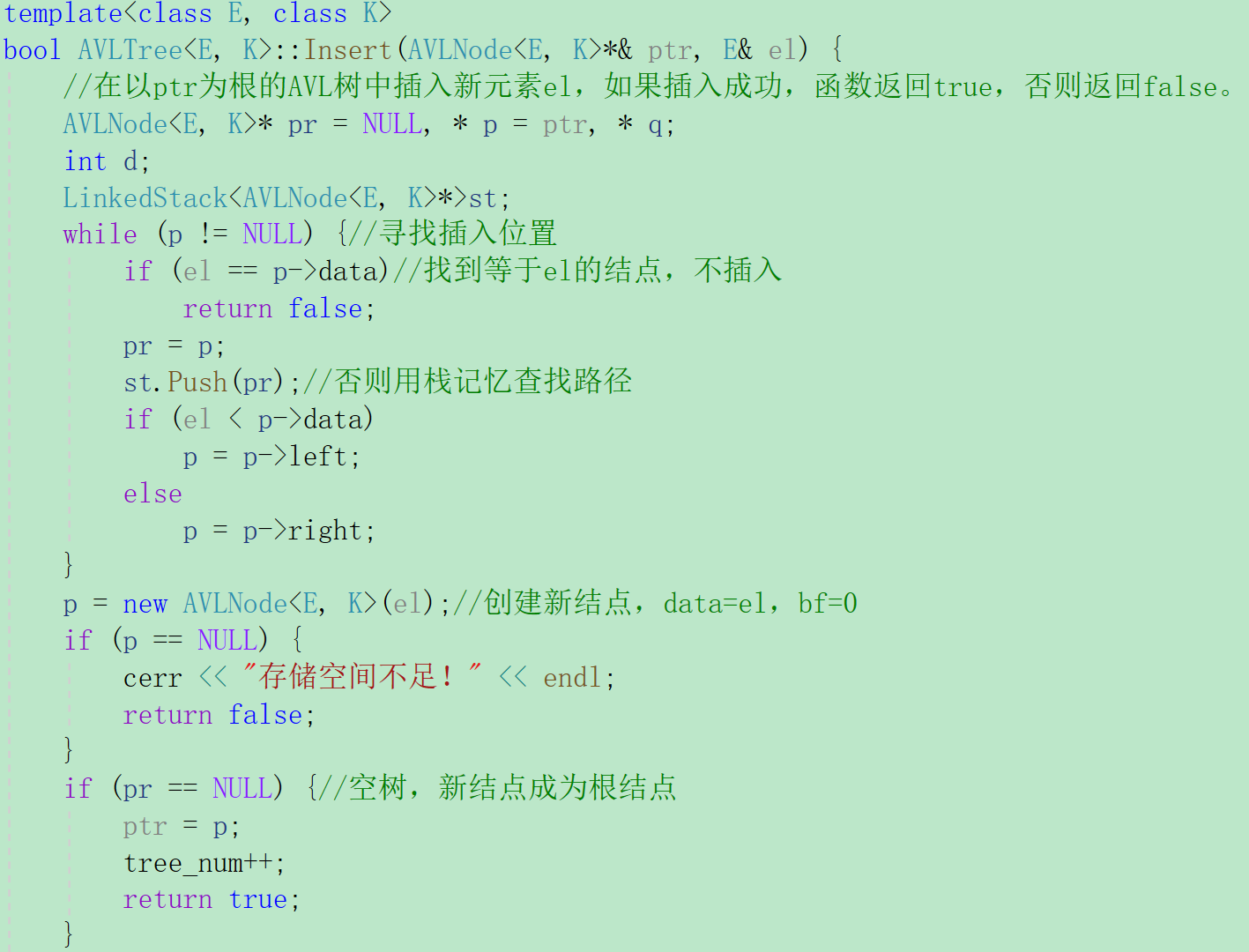
* bool Insert(AVLNode<E, K>\*& ptr, E& el);//插入，内部函数
* bool Remove(AVLNode<E, K>\*& ptr, K x);//删除，内部函数
* void RotateL(AVLNode<E, K>\*& ptr);//左单旋
* void RotateR(AVLNode<E, K>\*& ptr);//右单旋
* void RotateLR(AVLNode<E, K>\*& ptr);//先左单旋后右单旋
* void RotateRL(AVLNode<E, K>\*& ptr);//先右单旋后做单旋
* AVLNode<E, K>\* Search1(K x, AVLNode<E, K>\*& ptr);//搜索，外部接口
* AVLNode<E, K>\* find(K x) ;//查找，返回指向结点的指针，外部接口
* bool Insert(E& el) ;//插入，外部接口
* bool Remove(K x); //删除，外部接口
* AVLNode<E, K>\* get\_root(); //返回根结点
* int get\_Tnum() ;//返回树的结点数，用于统计
* void input(int i);//插入i个元素
* void ouput();//输出
* int count() ;//中序遍历计结点个数，统计的方法2
* friend ostream& operator<<(ostream& out, const AVLTree<E, K>& Tree) ;
* void Traverse(const AVLNode<E, K>\* ptr, ostream& out)const;//中序遍历输出
* void Traverse(const AVLNode<E, K>\* ptr, int& count)const;//中序遍历计数

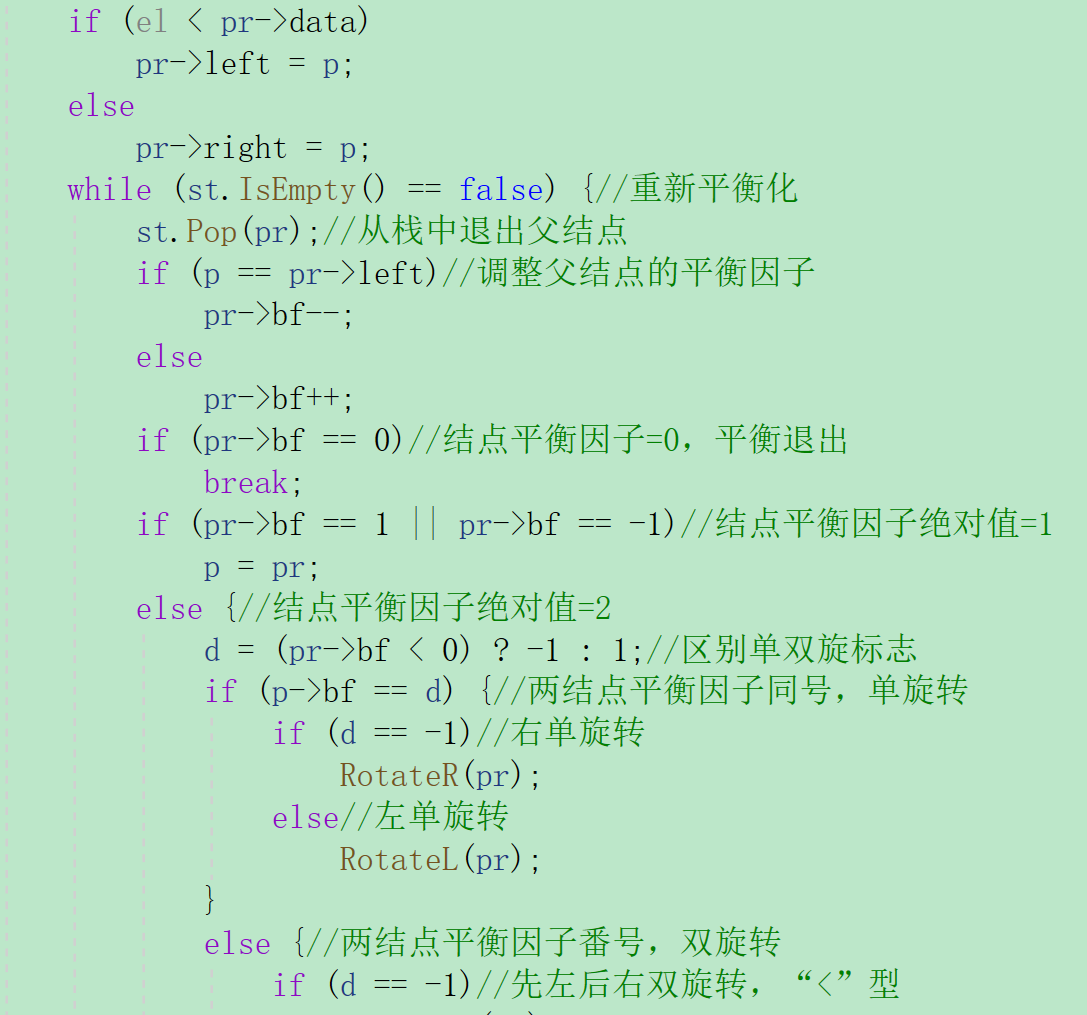
主要算法的时间复杂度：

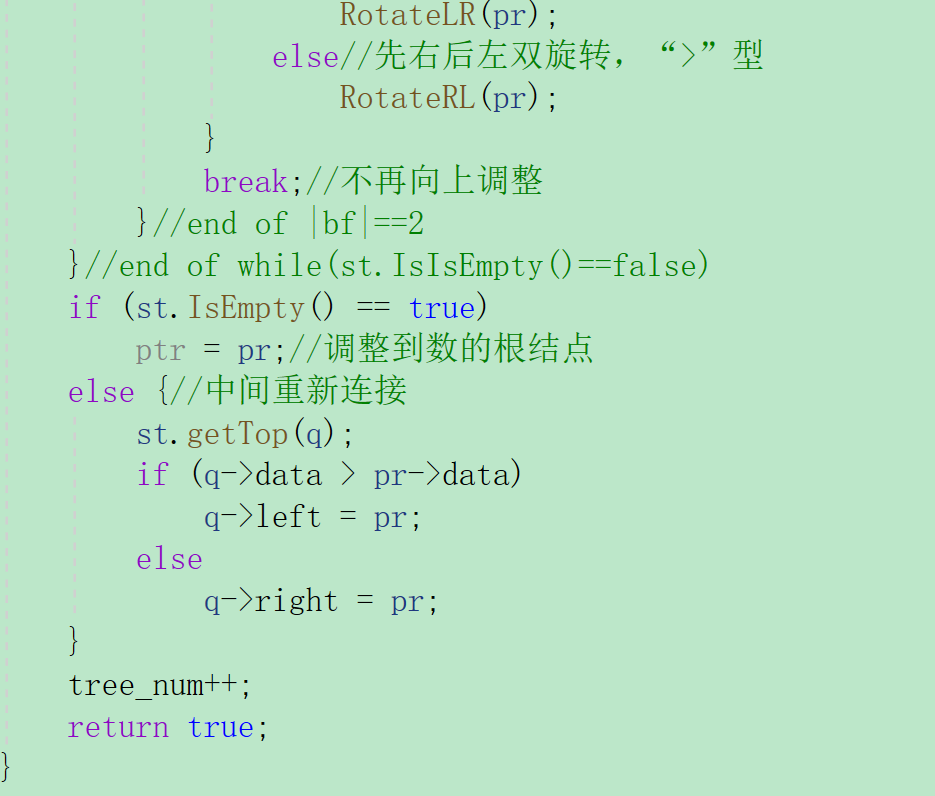
* 插入：O(log2n)
* 删除：O(log2n)
* 查找：O(log2n)
* 计数：O(n)
* 查找算法代码如下：



* 插入算法代码如下：

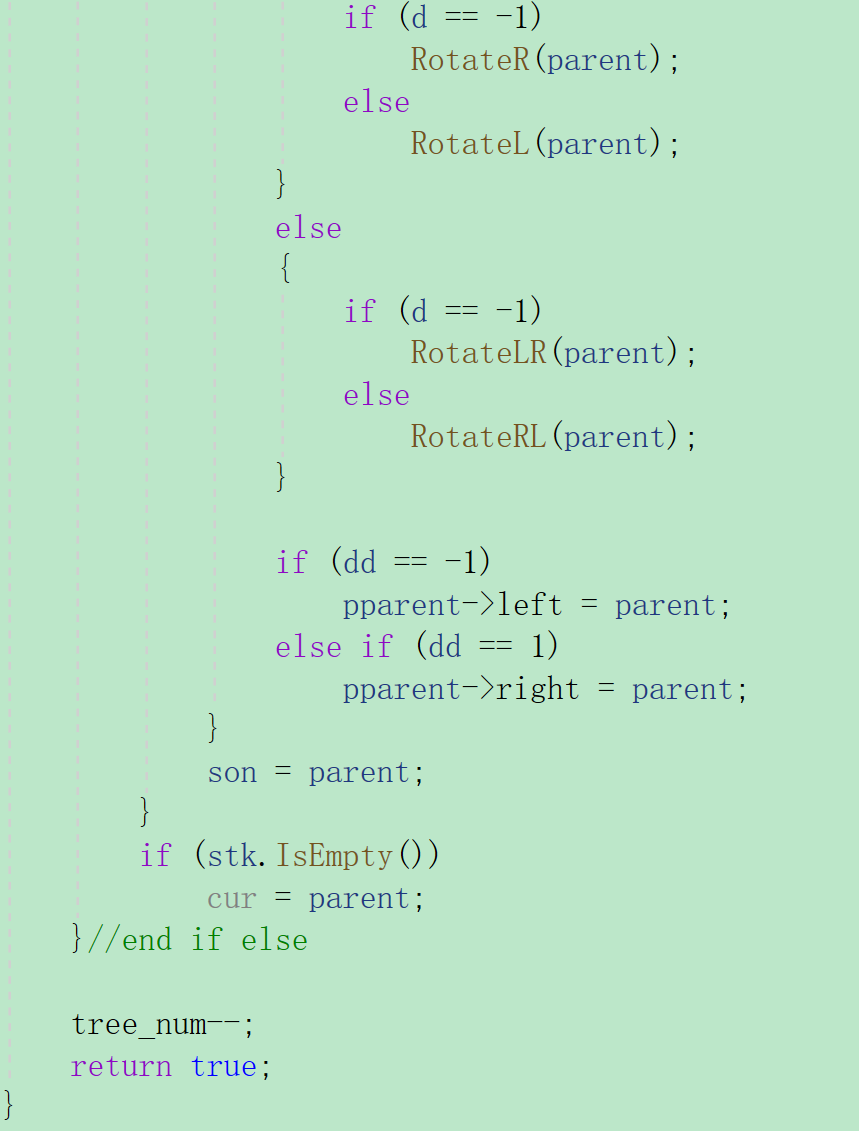
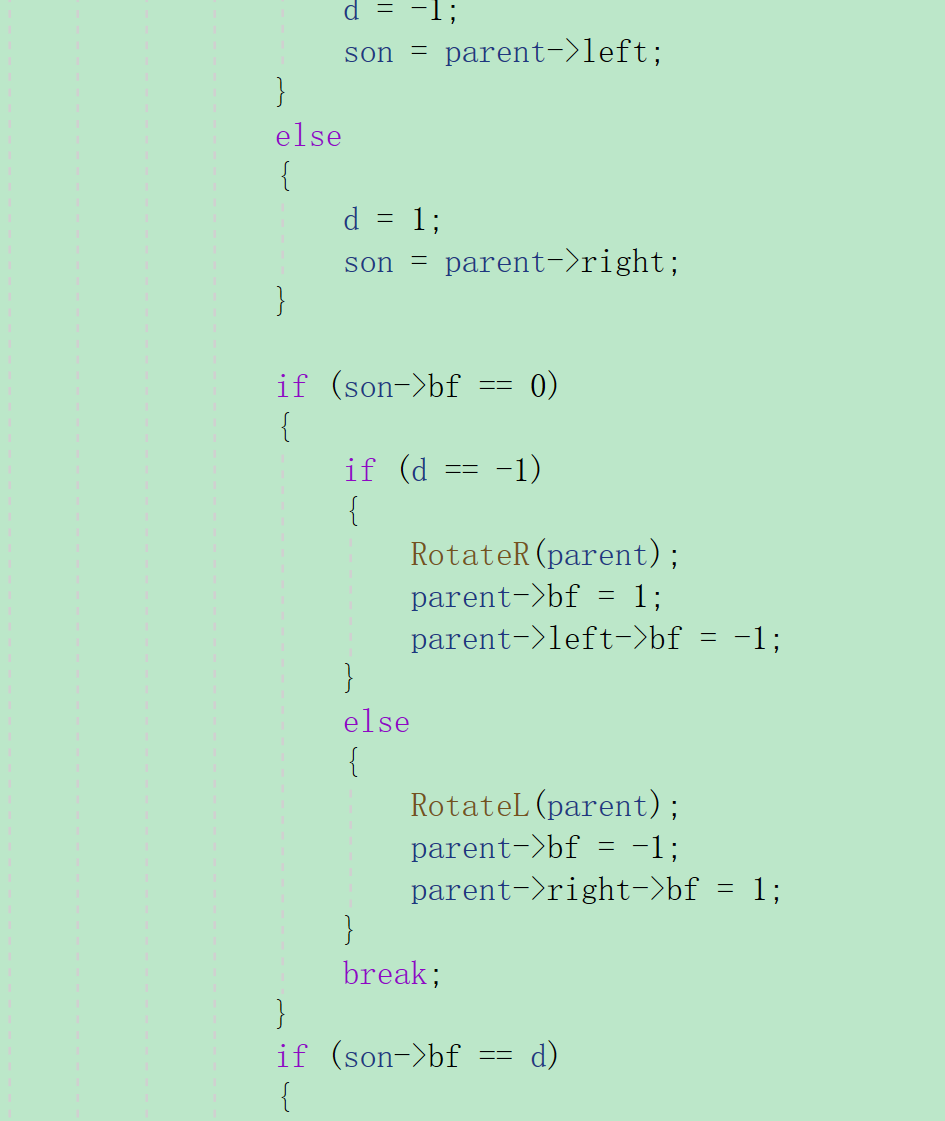
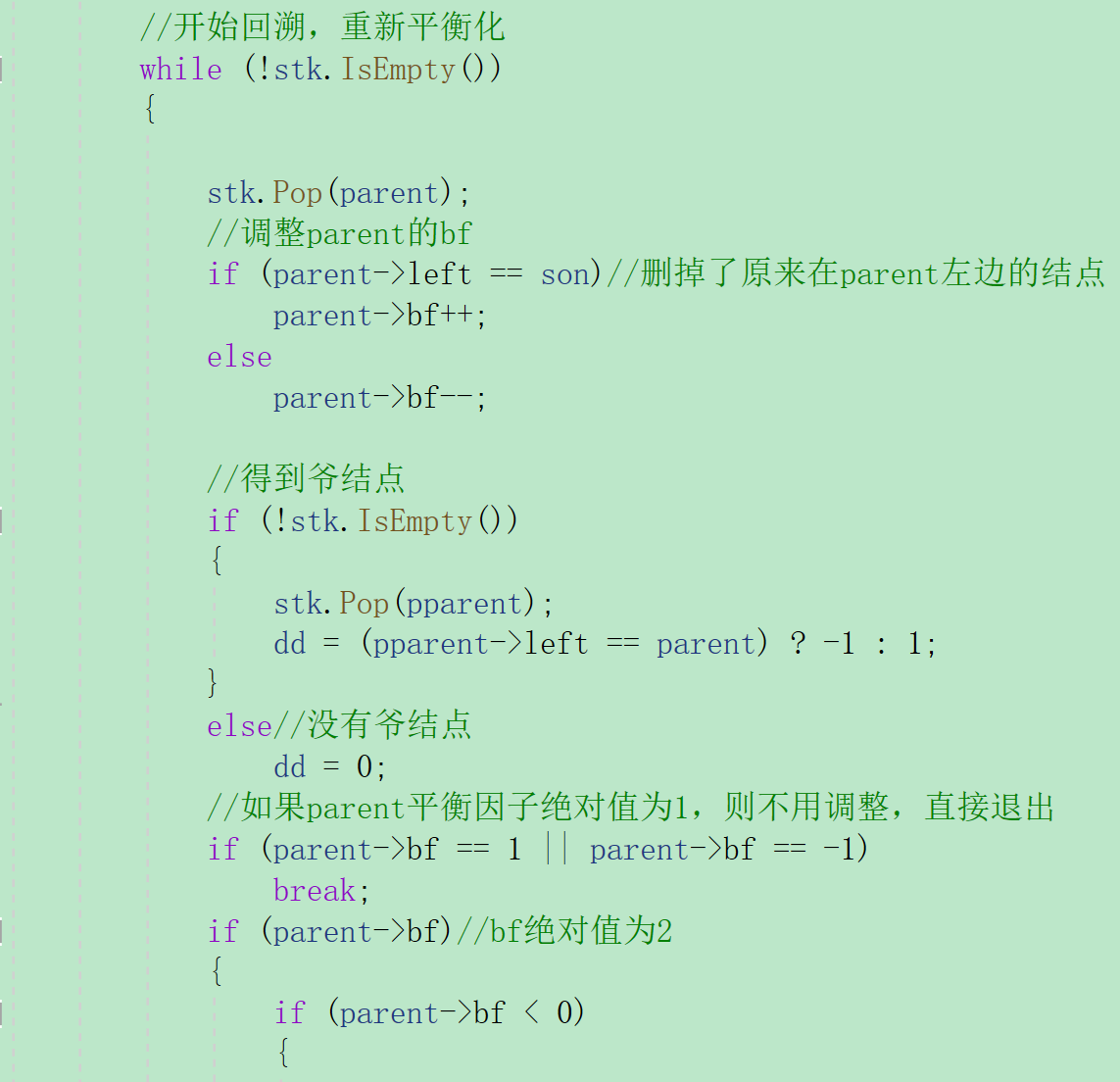
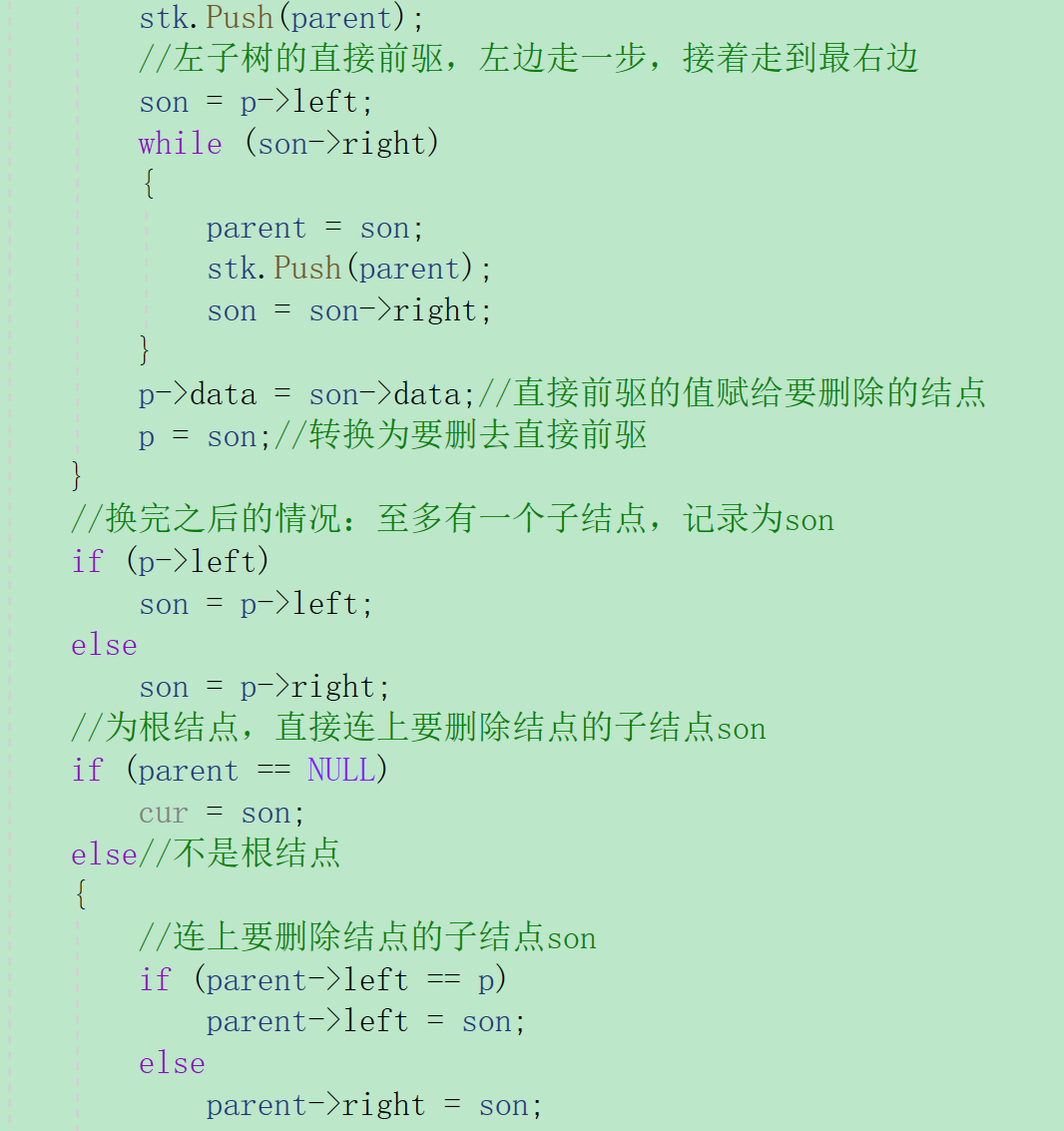






* 删除算法代码实现：





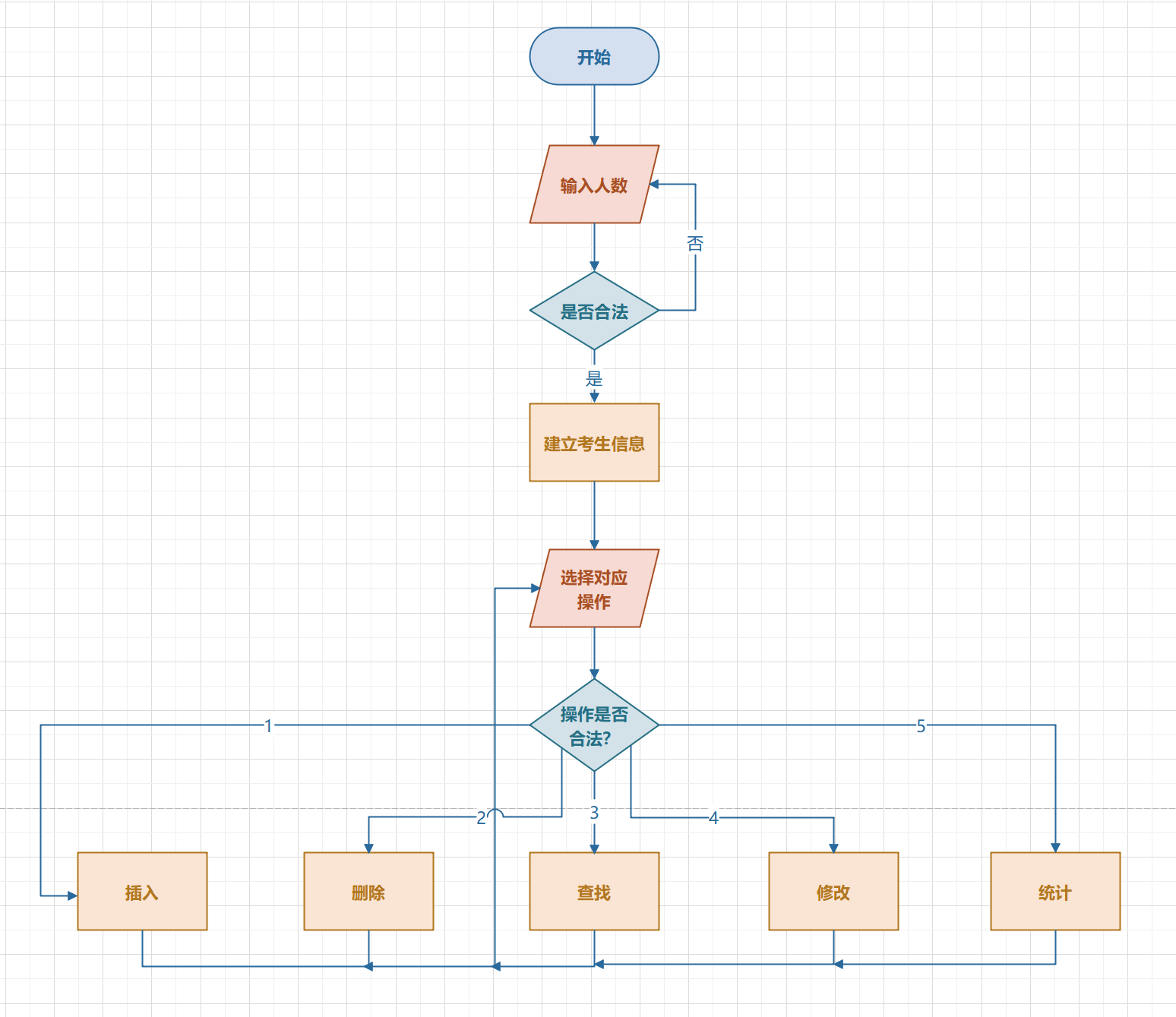
可以看出，与链表的插入、删除、查找算法相比，AVL树对于频繁需要查找结点的情况具有强大的优越性。

## 2.3 **项目算法**

### 2.3.1 实现思路

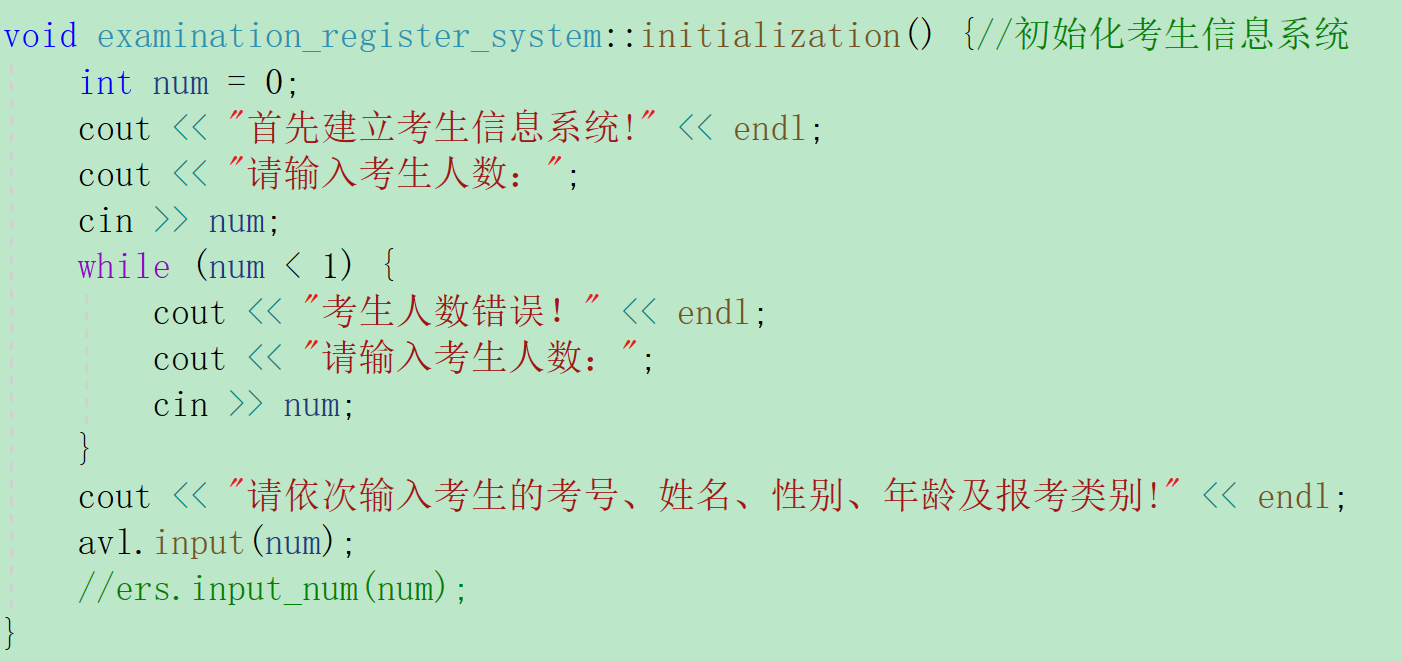
完成该项目的主要算法为考生信息的建立，查找，插入，修改，删除等功能，而建立、查找、插入、修改、删除功能归根到底本质为在一颗AVL树中搜索到某一位置后进行的不同操作。

总体流程图如下：

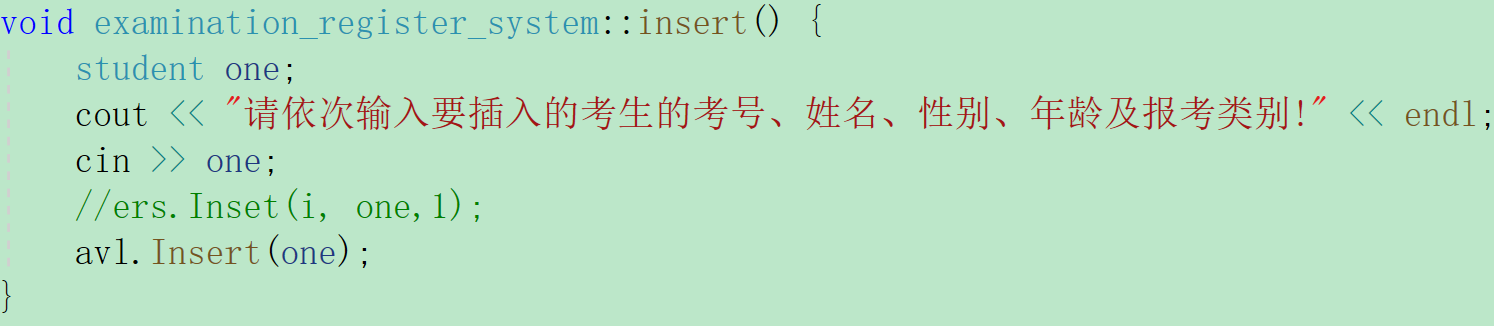


### 2.3.2 代**码实**现

1. 初始化考生信息系统：



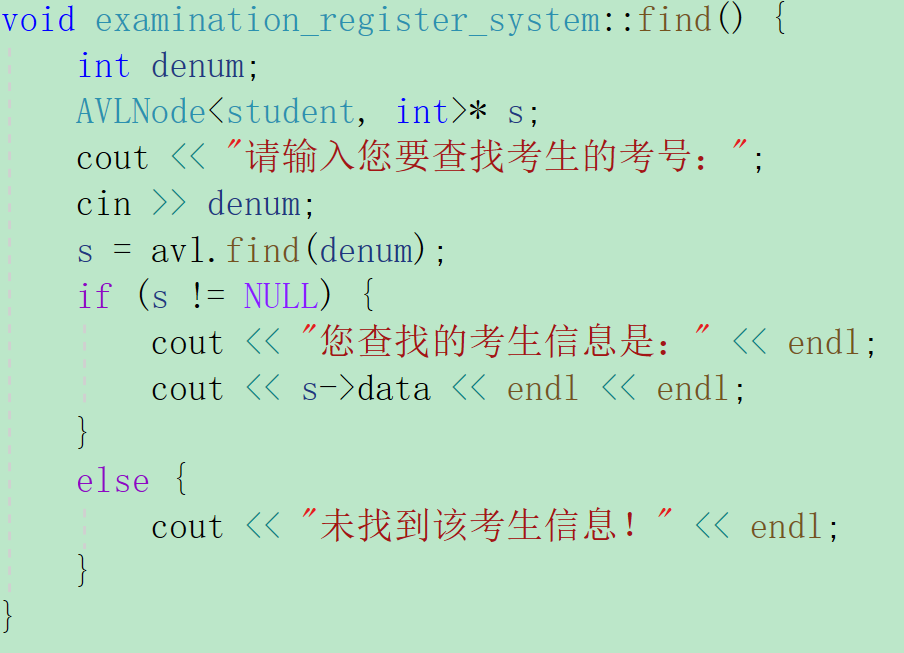
1. 考生信息插入：



1. 考生信息删除：



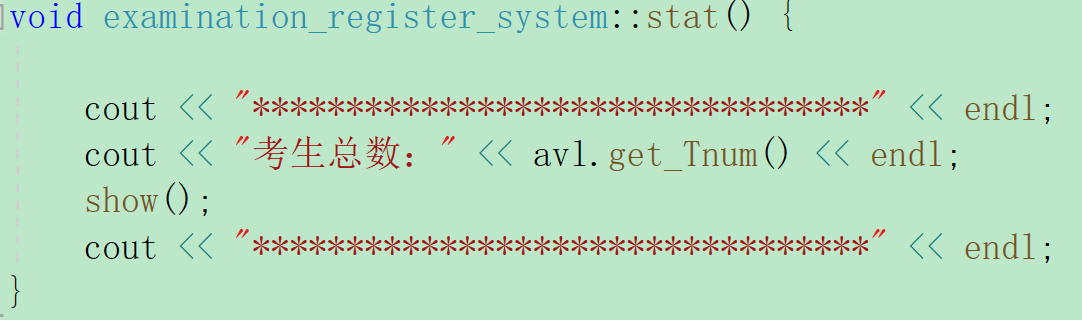
1. 考生信息查找：



1. 考生信息修改：



1. 考生人数统计及展示：



# 3 项目测试

## 3.1 **初始化考生信息**

测试输入：

4

2151299 苏家铭 男 20 软件工程

2151298 杨腾超 男 19 计算机

2151294 马威 男 21 土木工程

2151300 王大卫 女 18 电子信息

预期输出：

考号 姓名 性别 年龄 报考类别

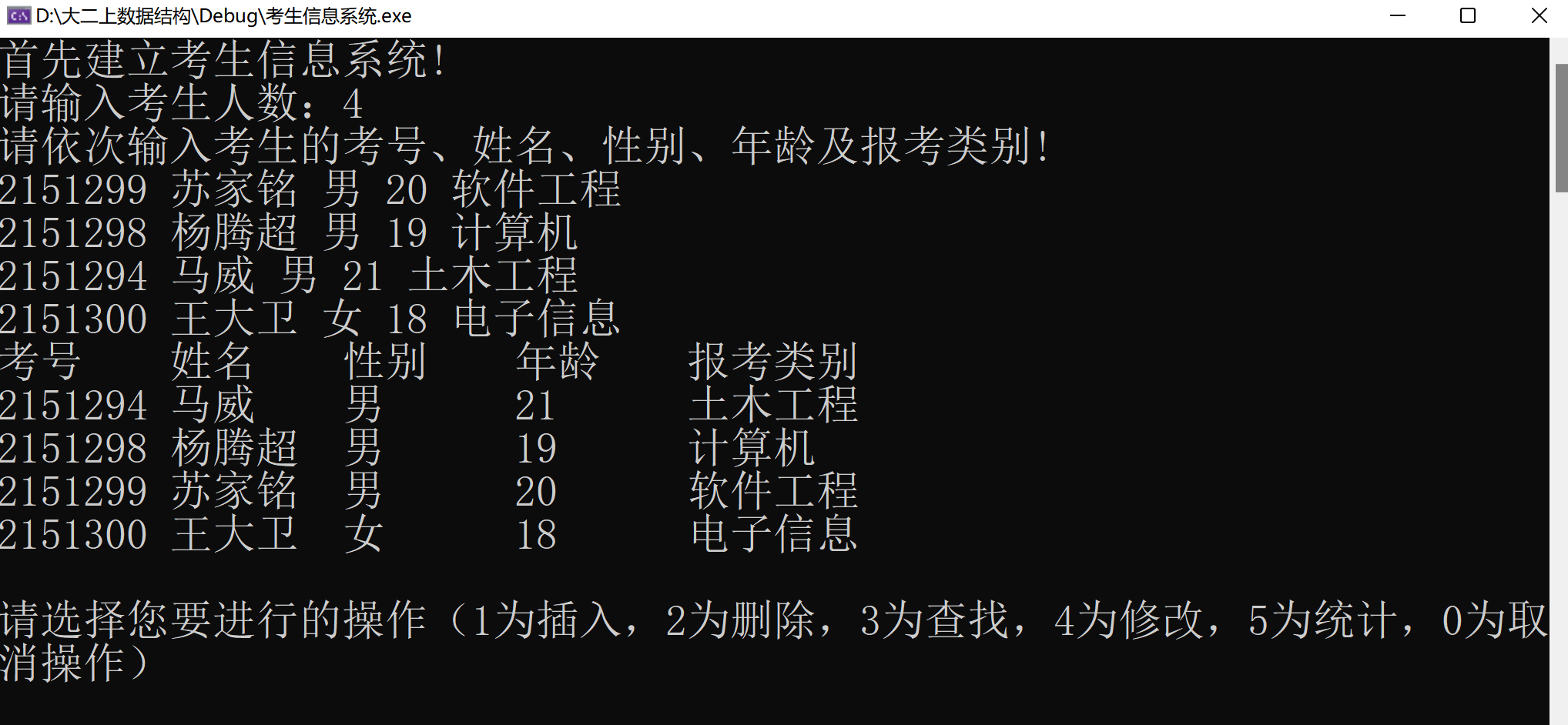
2151294 马威 男 21 土木工程

2151298 杨腾超 男 19 计算机

2151299 苏家铭 男 20 软件工程

2151300 王大卫 女 18 电子信息

实验结果：



## 3.2 插入考生信息

测试输入：

1

2151400 韦易燃 女 19 服装设计

预期输出：

考号 姓名 性别 年龄 报考类别

2151294 马威 男 21 土木工程

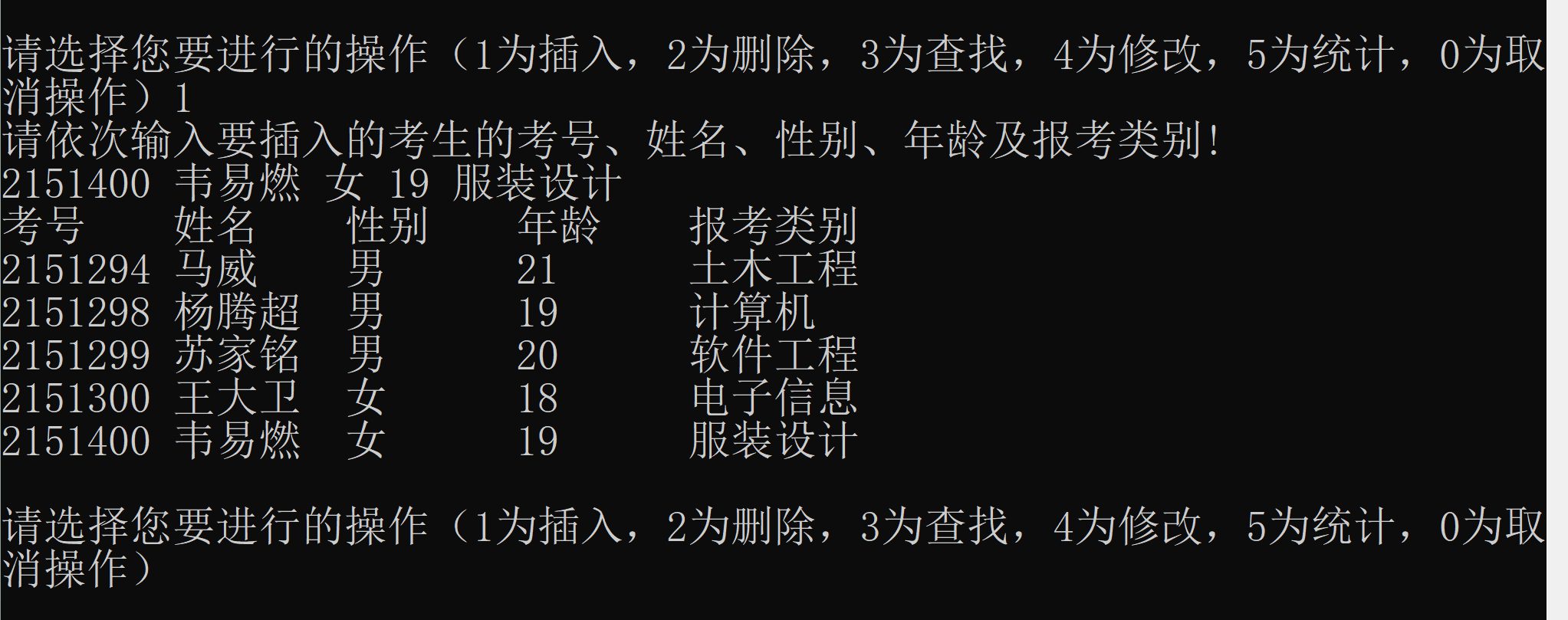
2151298 杨腾超 男 19 计算机

2151299 苏家铭 男 20 软件工程

2151300 王大卫 女 18 电子信息

2151400 韦易燃 女 19 服装设计

实验结果：



## 3.3 删除考生信息

测试输入：

2

2151298

预期输出：

您删除的考生信息是：

2151298 杨腾超 男 19 计算机

考号 姓名 性别 年龄 报考类别

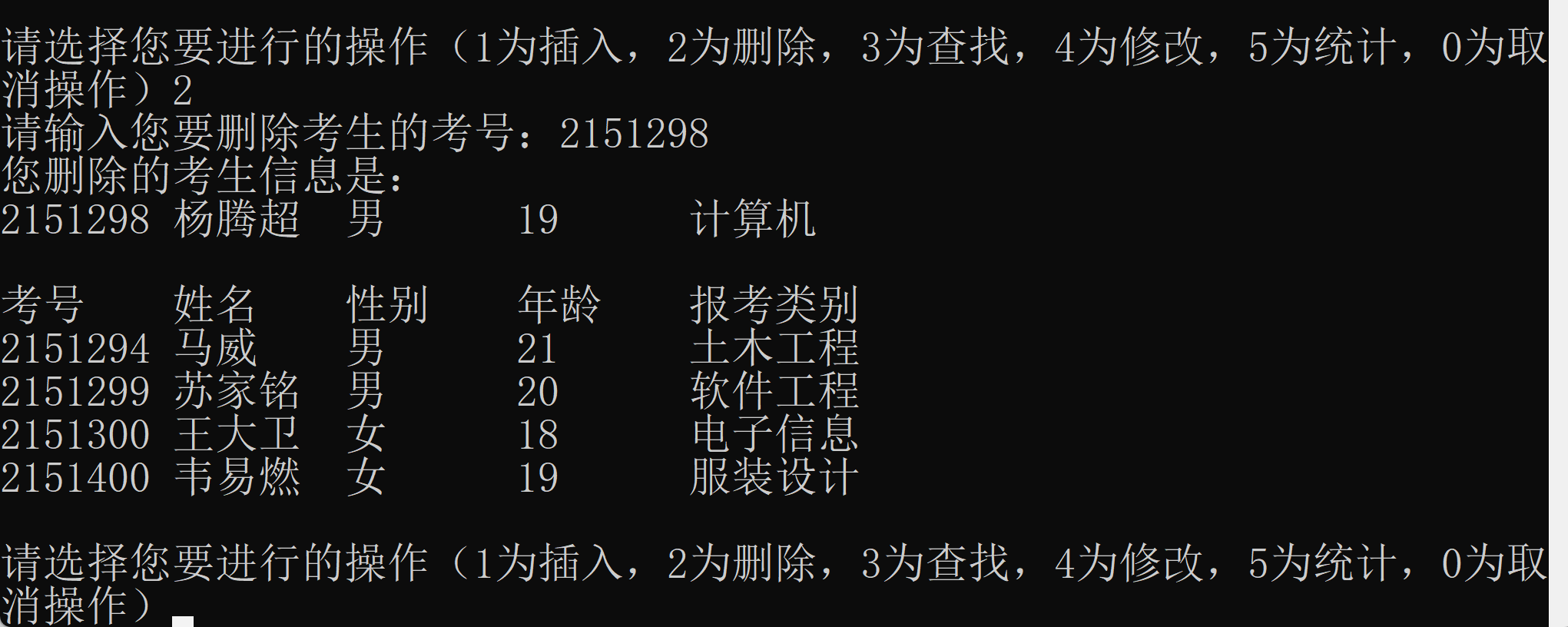
2151294 马威 男 21 土木工程

2151299 苏家铭 男 20 软件工程

2151300 王大卫 女 18 电子信息

2151400 韦易燃 女 19 服装设计

实验结果：



## 3.4 查找考生信息

测试输入：

3

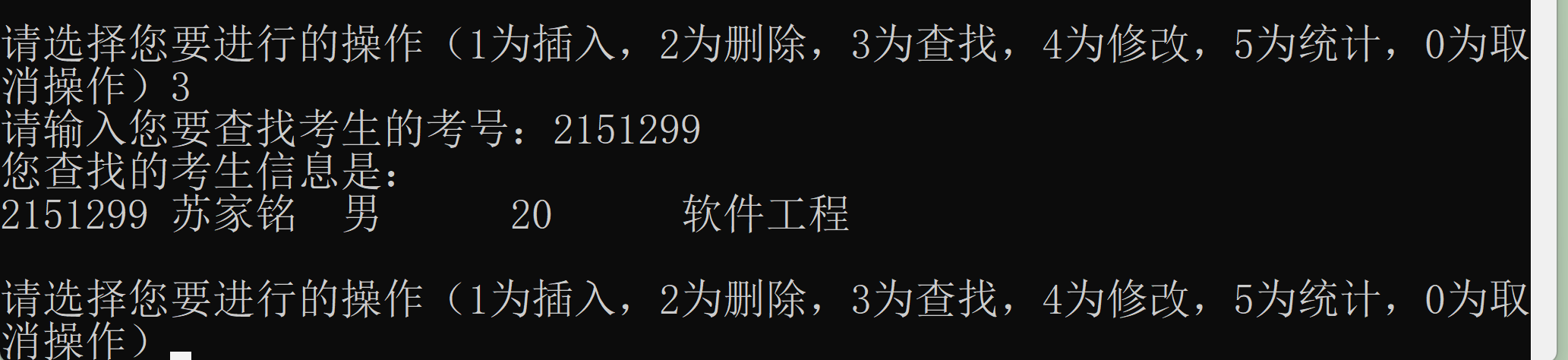
2151299

预期输出：

您查找的考生信息是：

2151299 苏家铭 男 20 软件工程

实验结果：



## 3.5 修改考生信息

测试输入：

4

2151294

预期输出：

您修改的考生信息是：

2151294 马威 男 21 土木工程

请输入修改后的考生考号、姓名、性别、年龄、报考类别！

测试输入：

2151296 马威威 男 22 土木环境工程

预期输出：

考号 姓名 性别 年龄 报考类别

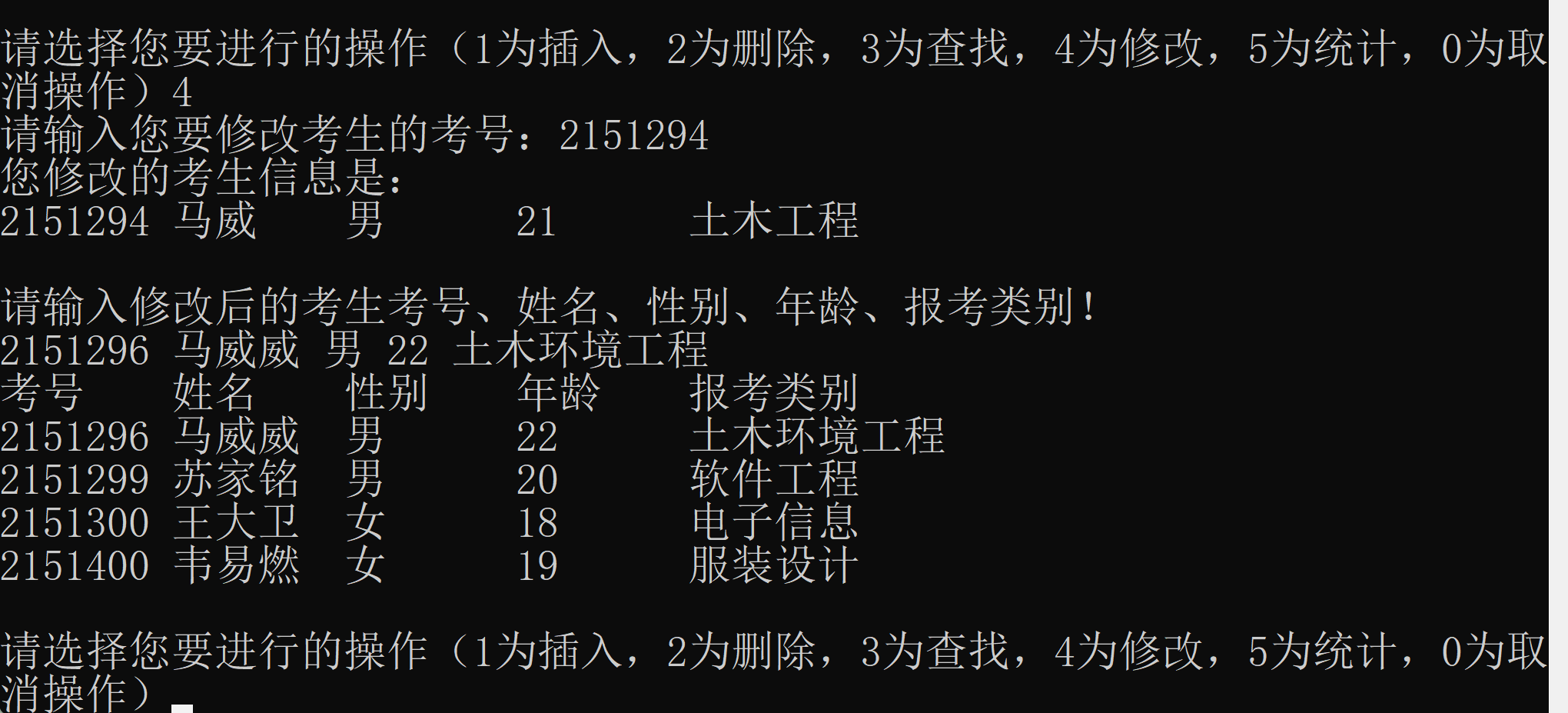
2151296 马威威 男 22 土木环境工程

2151299 苏家铭 男 20 软件工程

2151300 王大卫 女 18 电子信息

2151400 韦易燃 女 19 服装设计

实验结果：



## 3.6 统计考生数量并展示

测试输入：

5

预期输出：

考生总数：4

考号 姓名 性别 年龄 报考类别

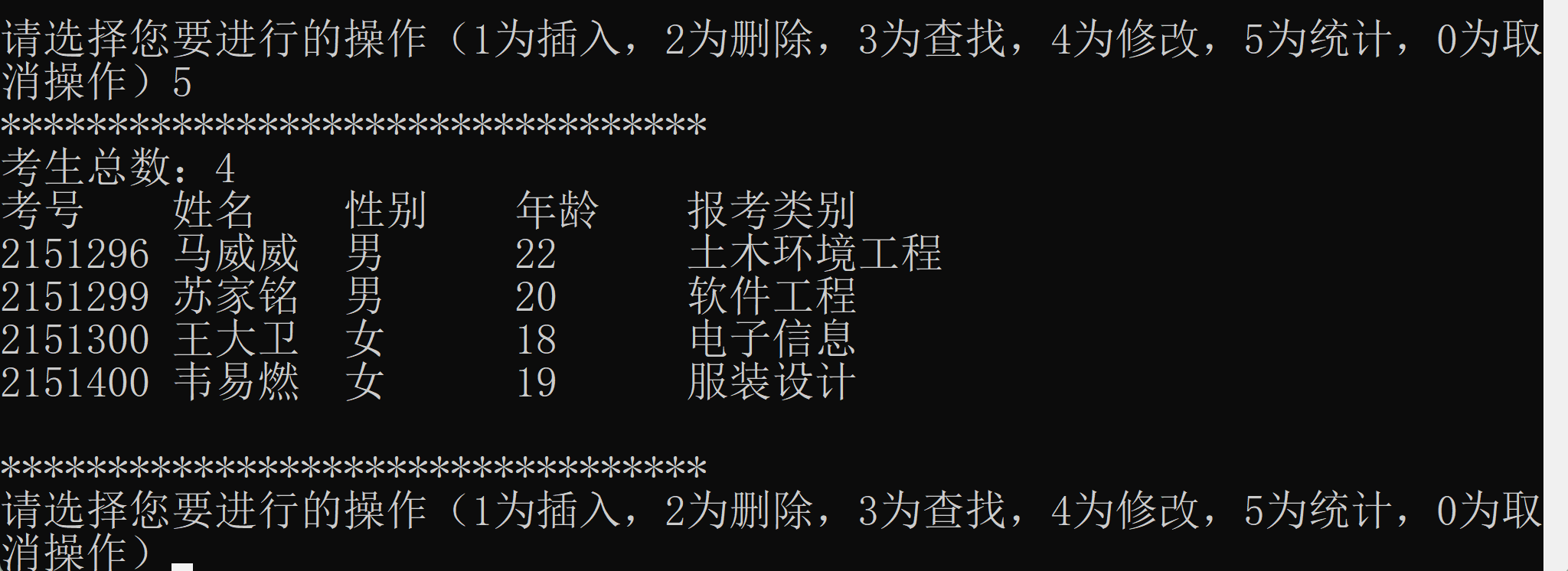
2151296 马威威 男 22 土木环境工程

2151299 苏家铭 男 20 软件工程

2151300 王大卫 女 18 电子信息

2151400 韦易燃 女 19 服装设计

实验结果：



## 3.7 退出系统

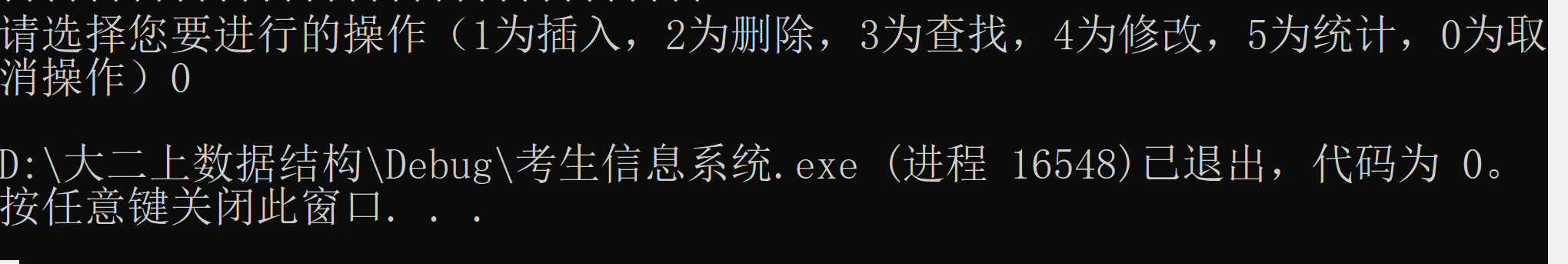
测试输入：

0

预期输出：

退出程序

实验结果：



## 3.8 边界测试1：初始化考生人数为负数

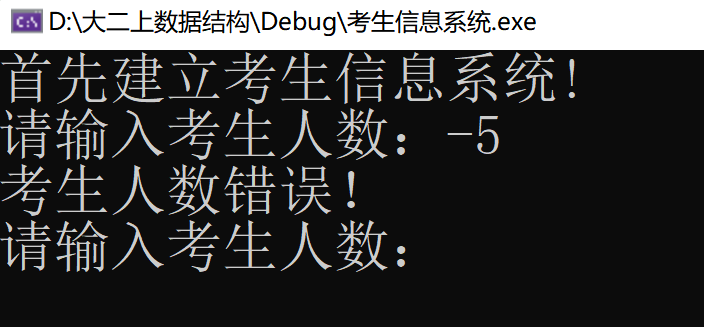
测试输入：

-5

预期输出：

考生人数错误！

实验结果：



## 3.9 边界测试2：输入考生信息错误（年龄或学号为负数）

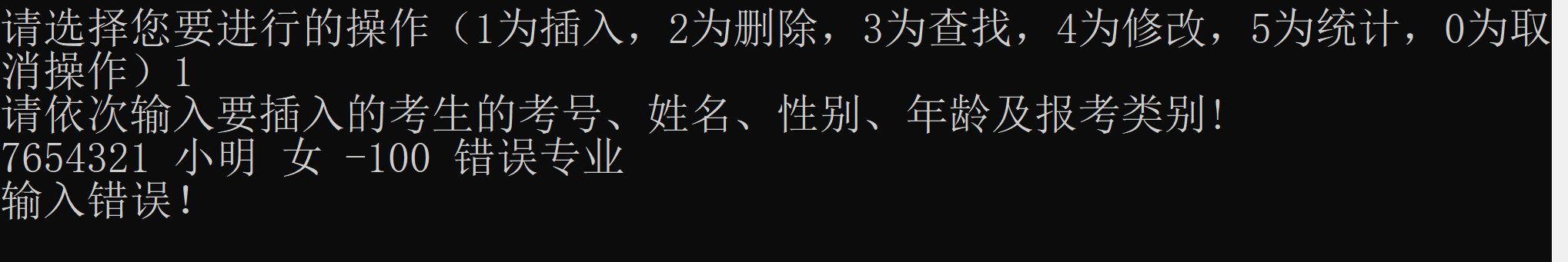
测试输入：

7654321 小明 女 -100 错误专业

预期输出：

输入错误！

实验结果：



## 3.10 边界测试3：未查找到考生信息

测试输入：

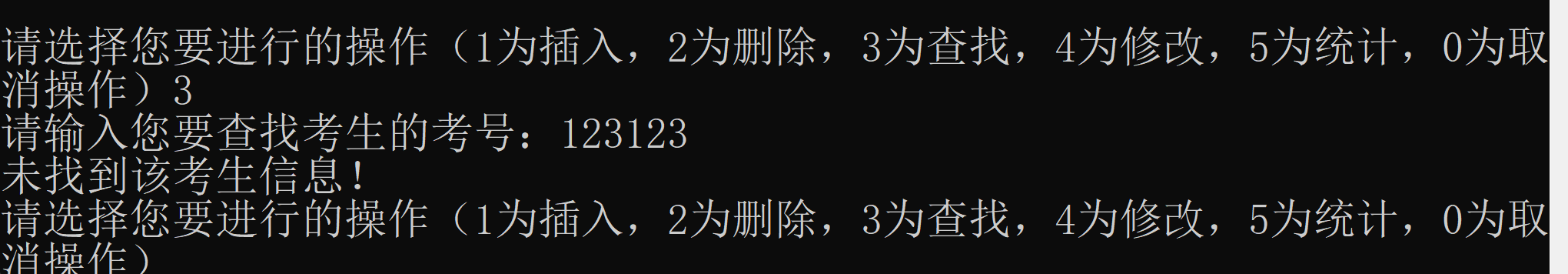
3

123123

预期输出：

未找到该考生信息！

实验结果：



## 3.11 边界测试4：删除不存在的考生信息

测试输入：

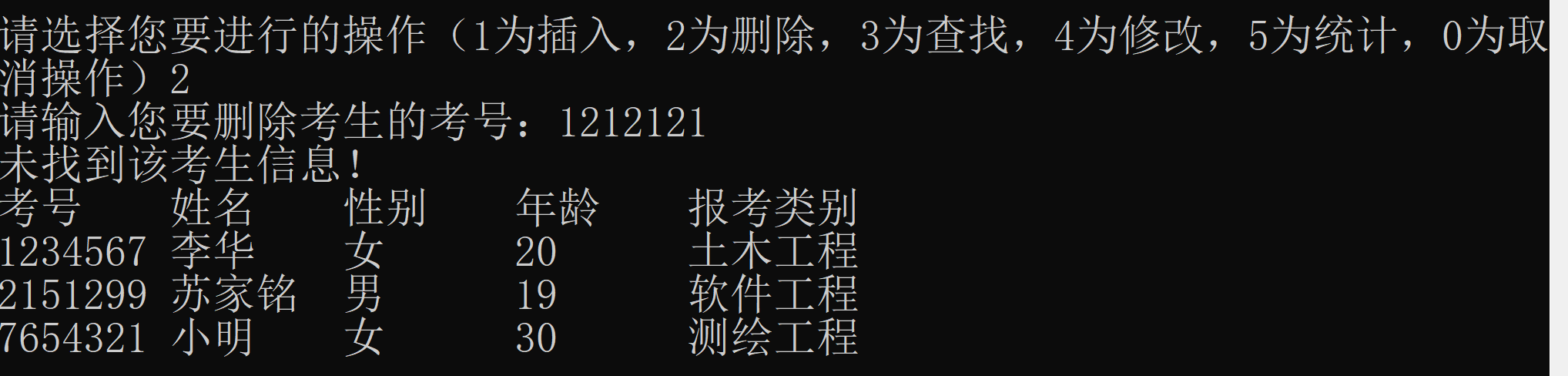
2

1212121

预期输出：

未找到该考生信息！

实验结果：



## 3.12 边界测试5：修改不存在的考生信息

测试输入：

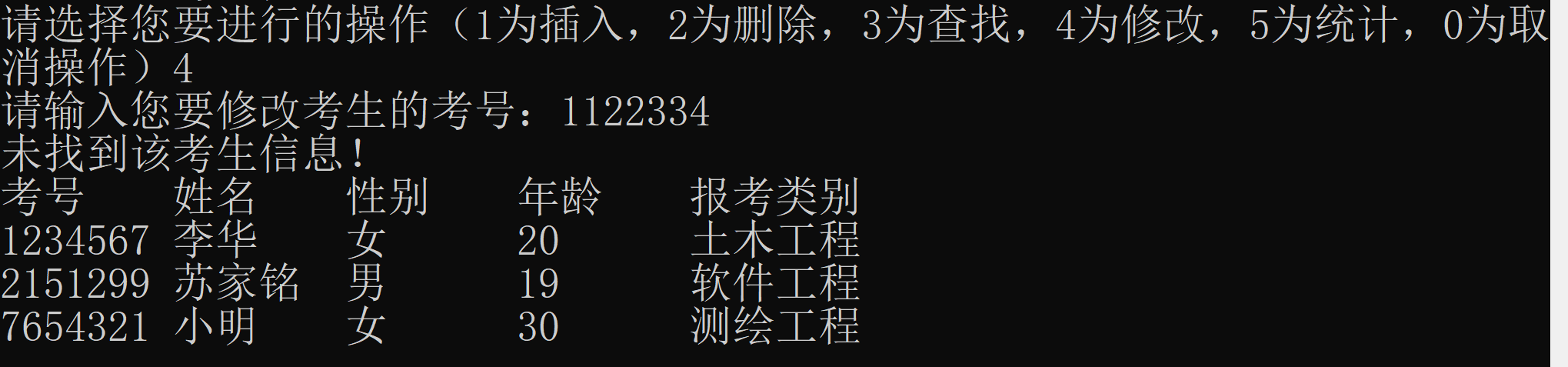
4

1122334

预期输出：

未找到该考生信息！

实验结果：



## 3.13 边界测试6：修改后的考生信息有误

测试输入：

4

1234567

54321 李华华 女 -20 土木工程

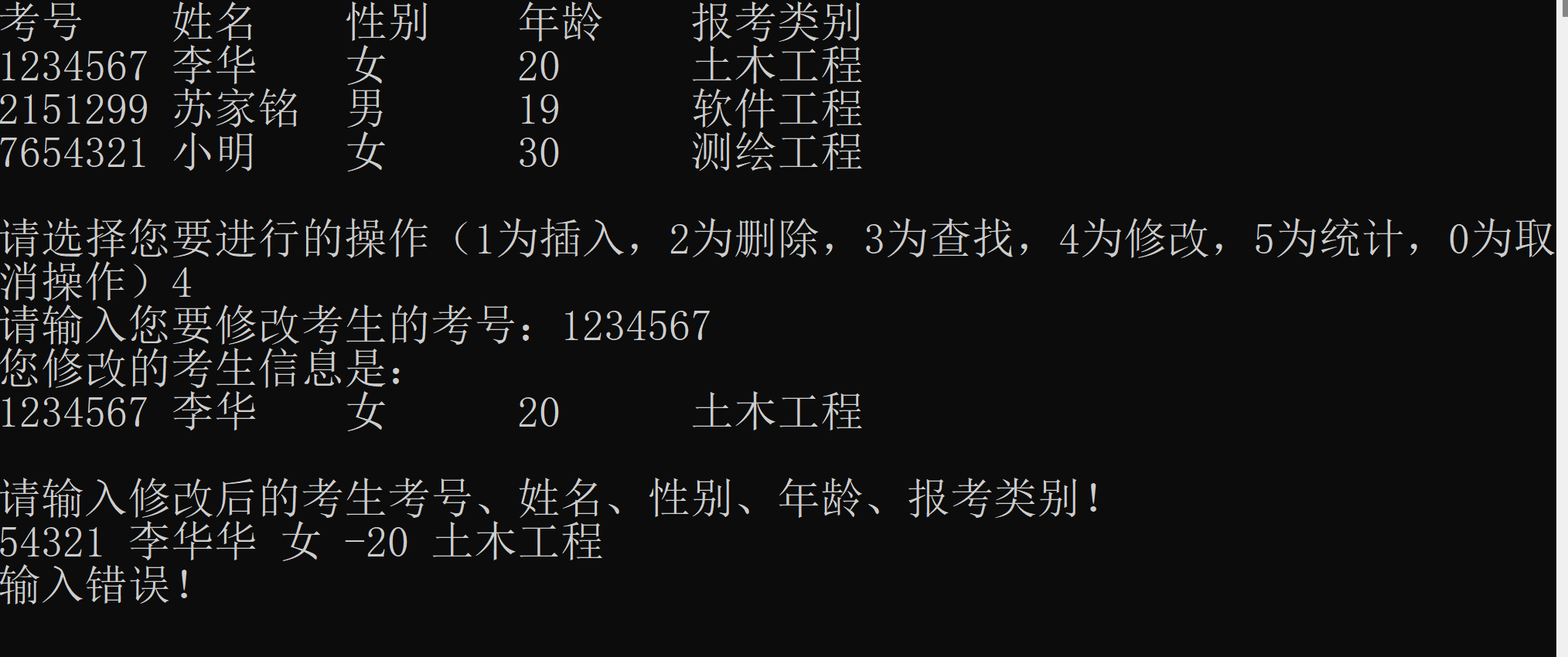
预期输出：

您修改的考生信息是：

1234567 李华 女 20 土木工程

输入错误！

实验结果：



## 3.14 边界测试7：选择操作非法

测试输入（非法数字）：

-1

预期输出：

输入的操作非法，请再次输入您要进行的操作：

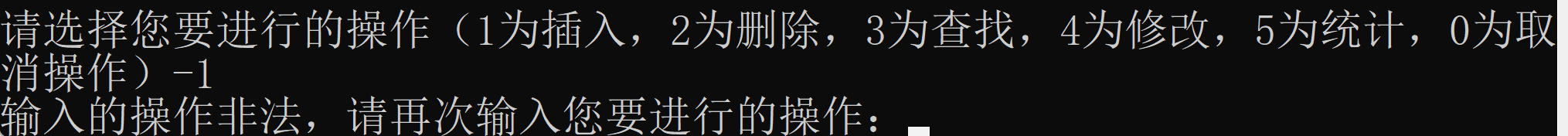
测试输入（非法字符）：

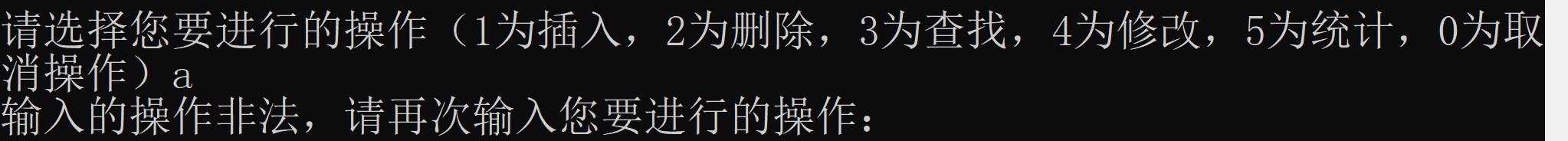
a

预期输出：

输入的操作非法，请再次输入您要进行的操作：

实验结果：





# 4 算法性能分析

## 4.1 正确性

本算法能正确地执行预定的功能和性能要求，由实验结果表明该AVL树实现的考生报名系统可以正确的实现考生信息的存取等多项功能。

## 4.2 可使用性

本算法可以很方便地使用，对考生信息的建立，查找，插入，修改，删除等功能已分别封装在一个特定的函数中。并且该算法有良好的界面和完备的用户文档。没有使用公用变量或全局变量。

## 4.3 可读性

本算法逻辑清晰、简单、且结构化，所有命名与函数名都具有实际含义，让人见名知义。且算法中包含了大量注释，简要说明了算法功能、输入与输出参数的使用规则、重要数据的作用、算法中各程序段完成的功能。

## 4.4 效率

链表对考生信息的建立，查找，插入，修改，删除等功能的时间复杂度为O(n),本项目使用AVL树来作为存储结构，使得算法时间复杂度降为O(log2n).

## 4.5 健壮性

本算法对于边界条件，诸如：初始化考生人数为负数、输入考生信息错误（年龄或学号为负数）、未查找到考生信息、删除不存在的考生信息、修改不存在的考生信息、修改后的考生信息有误、选择操作非法的情况都有相应的判断，并且对于AVL树中新结点的空间申请失败情况也有相应的错误提示。