项目说明文档

数据结构课程设计

——考试报名系统

作 者 姓 名： 陈翔飞

学 号： 1851756

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](#_Toc27342740)

[1.1 背景分析 4](#_Toc27342741)

[1.2 功能分析 4](#_Toc27342742)

[2 设计 4](#_Toc27342743)

[2.1 数据结构设计 4](#_Toc27342744)

[2.2 类结构设计 4](#_Toc27342745)

[2.3 成员与操作设计 5](#_Toc27342746)

[2.4 系统设计 11](#_Toc27342747)

[3 实现 12](#_Toc27342748)

[3.1 插入功能的实现 12](#_Toc27342749)

[3.1.1 插入功能流程图 12](#_Toc27342750)

[3.1.2 插入功能核心代码 12](#_Toc27342751)

[3.1.3 插入功能截屏示例 14](#_Toc27342752)

[3.2 删除功能的实现 15](#_Toc27342753)

[3.2.1 删除功能流程图 15](#_Toc27342754)

[3.2.2 删除功能核心代码 15](#_Toc27342755)

[3.2.3 删除功能截屏示例 18](#_Toc27342756)

[3.3 查找功能的实现 19](#_Toc27342757)

[3.3.1 查找功能流程图 19](#_Toc27342758)

[3.3.2 查找功能核心代码 19](#_Toc27342759)

[3.3.3 查找功能截图示例 20](#_Toc27342760)

[3.4 修改功能的实现 22](#_Toc27342761)

[3.4.1 修改功能流程图 22](#_Toc27342762)

[3.4.2 修改功能核心代码 22](#_Toc27342763)

[3.4.3 修改功能截屏示例 23](#_Toc27342764)

[3.5 统计功能的实现 25](#_Toc27342765)

[3.5.1 统计功能流程图 25](#_Toc27342766)

[3.5.2 统计功能核心代码 25](#_Toc27342767)

[3.5.3 统计功能截屏示例 26](#_Toc27342768)

[3.6 总体系统的实现 27](#_Toc27342769)

[3.6.1 总体系统流程图 27](#_Toc27342770)

[3.6.2 总体系统核心代码 27](#_Toc27342771)

[3.6.3 总体系统截屏示例 28](#_Toc27342772)

[4 测试 29](#_Toc27342773)

[4.1 功能测试 29](#_Toc27342774)

[4.1.1 插入功能测试 29](#_Toc27342775)

[4.1.2 删除功能测试 30](#_Toc27342776)

[4.1.3 查找功能测试 31](#_Toc27342777)

[4.1.4 修改功能测试 31](#_Toc27342778)

[4.1.5 统计功能测试 31](#_Toc27342779)

[4.2 边界测试 32](#_Toc27342780)

[4.2.1 初始化无输入数据 32](#_Toc27342781)

[4.2.2 删除根节点 32](#_Toc27342782)

[4.2.3 删除后AVL树为空 33](#_Toc27342783)

[4.2.4 插入考生考号极大 33](#_Toc27342784)

[4.2.5 删除考生考号极大 34](#_Toc27342785)

[4.2.6 查找考生考号极大 34](#_Toc27342786)

[4.2.7 修改考生考号极大 34](#_Toc27342787)

[4.3 出错测试 35](#_Toc27342788)

[4.3.1 考生人数错误 35](#_Toc27342789)

[4.3.2 操作码错误 35](#_Toc27342790)

[4.3.3 插入考生的考号已经存在 36](#_Toc27342791)

[4.3.4 删除考号不存在 36](#_Toc27342792)

[4.3.5 查找考号不存在 37](#_Toc27342793)

[4.3.6 修改考号不存在 37](#_Toc27342794)

# 1 分析

## 背景分析

在当今的教育环境下，考试报名系统在学校管理当中扮演着一个很重要的角色。但是随着学生数量的增多和需要管理的数据日益繁杂，手工进行这样规模的数据管理就很有可能出错，进而影响到学生利益和考试的正常进行。如何快速地处理大规模的数据以及频繁的查询，是当今考试报名系统需要解决的关键问题。

伴随着计算机技术的高速发展，相较于人工管理，使用计算机技术来进行考试信息的维护和管理，其具有管理容量上限高，管理快速，成本低等多个巨大优势，能极大提升考试报名管理的效率，有利于学校的发展。可以说，开发一个高效精准的考试报名系统是有必要，也有极大益处的。

## 1.2 功能分析

一个拥有基本功能的考试系统首先需要能将考生的对应信息进行输入并储存下来，同时其还应该能对已经储存的信息进行显示。其次，考试系统还应该能对已经储存的信息进行维护，能对考生信息进行插入，删除，修改，保证不存在废弃信息以及方便考生对自己的信息进行修正。最后，这个考试管理系统还应该可以正常关闭。

综上所述，该考试管理系统需要有输入、输出、插入、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，该系统要求大量的增加、删除、修改操作，常用的可用数据结构包括数组，链表，BST，考虑到需要对数据进行插入和删除，而数组在进行这种操作的时候很可能移动整个数组，太过耗费时间，而链表在进行查询操作时对于花费时间上界没有保证，于是综合考虑，该考试系统决定使用BST进行维护信息，但是单纯的BST也存在退化成链表的情况，于是考虑选用平衡二叉查找树，常见的平衡二叉查找树有很多，这里选用AVL平衡树，来保证所有操作（除了统计所有考生信息）的时间复杂度为O（）。

## 2.2 类结构设计

为了保证设计的数据结构的泛用性，本项目选择将TreeNode类和AVLTree类都设计为模板类，TreeNode类储存实际的键值对数据，AVLTree类储存树根，并提供插入、查找、删除、遍历等接口。为了让模板类能实际应用于本项目，还设计了BigInteger类、Student类以及StudentInformationSystem类，分别用于存储学号，存储考生信息，以及用于给用户提供实际的报名系统交互接口。

## 2.3 成员与操作设计

**树结点类（TreeNode）**

**类定义：**

1. **template** <**typename** Key, **typename** Value> **class** TreeNode
2. {
3. **public**:
4. **void** MidVisit(ostream& os);
5. TreeNode<Key, Value>() = **default**;
6. TreeNode<Key, Value>(Key K, Value V);
7. ~TreeNode<Key, Value>();
8. Key ElementKey;
9. Value ElementValue;
10. TreeNode<Key, Value>\* Left = nullptr, \*Right = nullptr;
11. **int** Height = 0;
12. };

**公有成员：**

Key ElementKey；//树节点的键，作为平衡树的关键词

Value ElementValue；//树节点的值，储存需要保存的信息

TreeNode<Key,Value>\* Left；//树节点的左儿子

TreeNode<Key,Value>\* Right；//树节点的右儿子

Int Height；//树节点的高度，用于维护平衡

**公有操作：**

TreeNode<Key, Value>() = default; //默认构造函数

TreeNode<Key, Value>(Key K, Value V); //含键值对参数的构造函数

void MidVisit(ostream& os);//中序遍历该节点并输出值

**AVL树类（AVLTree）**

**类定义：**

1. **template** <**typename** Key = **int**, **typename** Value = **int**> **class** AVLTree
2. {
4. **public**:
5. AVLTree<Key, Value>();
6. AVLTree<Key, Value>(Key K, Value V);
7. ~AVLTree<Key, Value>();
8. TreeNode<Key, Value>\* Change(Key K, Value V);
9. TreeNode<Key, Value>\* Insert(Key K, Value V = Value());
10. TreeNode<Key, Value>\* Find(Key K);
11. TreeNode<Key, Value>\* Delete(Key K);
12. **void** MakeEmpty();
13. **void** DisPlay(ostream& os);
14. **int** GetHeight();
15. TreeNode<Key, Value>\* UncheckChange(TreeNode<Key, Value>\* P, Value V);
16. TreeNode<Key, Value>\* UncheckInsert(Key K, Value V);
17. **int** GetSize();
18. **private**:
19. **int**  Max(**int** x, **int** y);
20. **int** GetHeight(TreeNode<Key, Value>\* P);
21. TreeNode<Key, Value>\* SingleRotateWithLeft(TreeNode<Key, Value>\* K2);
22. TreeNode<Key, Value>\* SingleRotateWithRight(TreeNode<Key, Value>\* K2);
23. TreeNode<Key, Value>\* DoubleRotateWithLeft(TreeNode<Key, Value>\* K3);
24. TreeNode<Key, Value>\* DoubleRotateWithRight(TreeNode<Key, Value>\* K3);
25. TreeNode<Key, Value>\* Find(Key K, TreeNode<Key, Value>\* P);
26. TreeNode<Key, Value>\* FindMin(TreeNode<Key, Value>\* P);
27. TreeNode<Key, Value>\* FindMax(TreeNode<Key, Value>\* P);
28. TreeNode<Key, Value>\* Insert(Key K, Value V, TreeNode<Key, Value>\* P);
29. TreeNode<Key, Value>\* Delete(Key K, TreeNode<Key, Value>\* P);
30. TreeNode<Key, Value>\* MakeEmpty(TreeNode<Key, Value>\* P);
31. **int** Size = 0;
32. TreeNode<Key, Value>\* Root = nullptr;
33. };

**私有成员：**

int Size = 0;//AVL树的大小，即储存键值对数

TreeNode<Key, Value>\* Root ;//AVL树的树根

**私有操作：**

int Max(int x, int y); //返回x，y中的最大值

int GetHeight(TreeNode<Key, Value>\* P); //获取节点P的高度

TreeNode<Key, Value>\* SingleRotateWithLeft(TreeNode<Key, Value>\* K2);

//左单旋，用于维护平衡

TreeNode<Key, Value>\* SingleRotateWithRight(TreeNode<Key, Value>\* K2);

//右单旋，用于维护平衡

TreeNode<Key, Value>\* DoubleRotateWithLeft(TreeNode<Key, Value>\* K3);

//左双旋，用于维护平衡

TreeNode<Key, Value>\* DoubleRotateWithRight(TreeNode<Key, Value>\* K3);

//右双旋，用于维护平衡

TreeNode<Key, Value>\* Find(Key K, TreeNode<Key, Value>\* P);

//以节点P为根，查找有关键词K的节点

TreeNode<Key, Value>\* FindMin(TreeNode<Key, Value>\* P);

//以节点P为根，查找最小节点

TreeNode<Key, Value>\* FindMax(TreeNode<Key, Value>\* P);

//以节点P为根，查找最大节点

TreeNode<Key, Value>\* Insert(Key K, Value V, TreeNode<Key, Value>\* P);

//以节点P为根，插入键值对<K,V>

TreeNode<Key, Value>\* Delete(Key K, TreeNode<Key, Value>\* P);

//以节点P为根，删除有关键词K的节点

TreeNode<Key, Value>\* MakeEmpty(TreeNode<Key, Value>\* P);

//清空节点P以及P的所有子节点并释放内存

**公有操作：**

AVLTree<Key, Value>();

//AVLTree的默认构造函数

AVLTree<Key, Value>(Key K, Value V);

//AVLTree的带键值对参数的构造函数，以参数中的键值对开辟一个树节点并使Root指向该节点

~AVLTree<Key, Value>();

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

TreeNode<Key, Value>\* Change(Key K, Value V);

//将树中以K为键的节点的值改为V，如果不存在就插入键值对<K,V>

TreeNode<Key, Value>\* Insert(Key K, Value V = Value());

//将键值对<K,V>插入树中，如果键K已经存在则将对应节点中的值改为V

TreeNode<Key, Value>\* Find(Key K);

//返回键为K的树节点指针

TreeNode<Key, Value>\* Delete(Key K);

//删除键为K的树节点

void MakeEmpty();

//清空整个树并释放内存

void DisPlay(ostream& os);

//中序展示整棵树的所有树节点

int GetHeight();

//返回树的高度

TreeNode<Key, Value>\* UncheckChange(TreeNode<Key, Value>\* P, Value V);

//直接将节点P的值改为V

TreeNode<Key, Value>\* UncheckInsert(Key K, Value V);

//不检查K是否存在于树中，直接插入键值对<K,V>，如果插入失败则返回空

int GetSize();

//返回已经储存的键值对数量

**大整数类（BigInteger）**

**类定义：**

1. **class** BigInteger
2. {
3. **public**:
4. BigInteger(string Integer = {}) :Integer(Integer) {};
5. string Integer = {};
6. **bool** operator > (**const** BigInteger B) **const**;
7. **bool** operator < (**const** BigInteger B) **const**;
8. **bool** operator ==(**const** BigInteger B) **const**;
10. BigInteger& operator =(string s);
11. **friend** ostream &operator <<(ostream &os, **const** BigInteger &B);
12. **friend** istream& operator >>(istream& is, BigInteger &B);
13. };

**公有成员：**

string Integer ;//用字符串储存的整数

**公有操作：**

BigInteger(string Integer = {}) :Integer(Integer) {};

//带默认参数的构造函数

bool operator > (const BigInteger B) const;

//重载>运算符使该类可以进行>运算

bool operator < (const BigInteger B) const;

//重载<运算符使该类可以进行<运算

bool operator ==(const BigInteger B) const;

//重载==运算符使该类可以进行==运算

BigInteger& operator =(string s);

//重载=运算符使该类可以进行赋值运算

friend ostream &operator <<(ostream &os, const BigInteger &B);

//重载<<运算符使该类可以进行<<运算

friend istream& operator >>(istream& is, BigInteger &B);

//重载>>运算符使该类可以进行>>运算

**学生类（Student）**

**类定义：**

1. **class** Student
2. {
4. **public**:
5. **static** **const** **int**  LineLength = 12;
6. **static** string  Display(**const** string & s, **int** Delta = 0);
7. Student() = **default**;
8. Student(string ID, string N, string S, string A, string T);
9. Student& operator =(Student S)
10. **friend** ostream & operator<<(ostream & os, **const** Student & s);
11. **friend** istream& operator >>(istream& is, **const**  Student &S);
12. BigInteger ID, Age;
13. string Sex, Name, TestCatagory;

16. };

**公有成员：**

BigInteger ID;

//学生学号

BigInteger Age;

//学生年龄

string Sex;

//学生性别

string Name,

//学生姓名;

string TestCatagory;

//考试类别

static const int LineLength = 12;

//行长度，用于格式化输出

**公有操作：**

static string Display(const string & s, int Delta = 0);

//格式化输出

Student() = default;

//默认构造函数

Student(string ID, string N, string S, string A, string T);

//带参数的构造函数

Student& operator =(Student S);

//重载=运算符，使该类支持赋值操作

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

//重载<<运算符使该类可以进行<<运算

friend istream& operator >>(istream& is, const Student &S);

//重载>>运算符使该类可以进行>>运算

**学生信息系统类（StudentInformationSystem）**

**类定义：**

1. **class** StudentInformationSystem
2. {
3. **public**:
4. StudentInformationSystem(){}
5. ~StudentInformationSystem(){}
6. **void** Init();
7. **void** Insert();
8. **void** Change();
9. **void** Delete();
10. **void** Find();
11. **void** Display();
12. **int** GetStudentNum();
13. **private**:
14. Student HeadLine;
15. AVLTree<BigInteger, Student> StudentTree;
16. };

**私有成员：**

Student HeadLine;

//方便格式化输出的标题线

AVLTree<BigInteger, Student> StudentTree;

//存储考生信息的AVLTree，键为考号

**公有操作：**

StudentInformationSystem(){}

//默认构造函数

~StudentInformationSystem(){}

//析构函数

void Init();

//初始化函数，用于屏幕初始化并完成初始学生的输入

void Insert();

//学生信息插入函数

void Change();

//学生信息修改函数

void Delete();

//学生信息删除函数

void Find();

//学生信息查询函数

void Display();

//统计函数

int GetStudentNum();

//获取学生数量

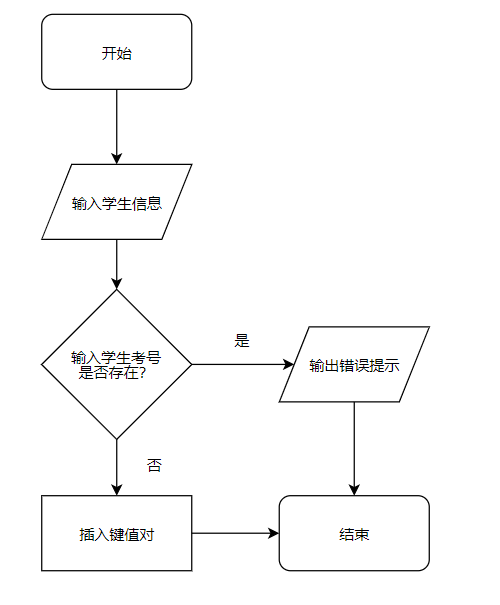
## 2.4 系统设计

系统首先调用Init ()函数实现对屏幕的初始化，完成输入数据工作，然后根据用户所输入的操作码（Command）执行AVL树对应的成员函数。

# 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

**用户界面接口：**

1. **inline** **void** StudentInformationSystem::Insert()
2. {
3. cout << "请依次输入要插入考生的考号，姓名，性别，年龄及报考类别：" << endl;
4. Student TempStudent;
5. cin >> TempStudent;
6. auto FoundPtr = StudentTree.Find(TempStudent.ID);
7. **if** (FoundPtr)
8. {
9. cout << "该学号已被其他考生使用，请检查插入考生信息是否错误" << endl;
10. **return**;
11. }
12. **else**
13. {
14. StudentTree.UncheckInsert(TempStudent.ID, TempStudent);
15. cout << "插入完成!" << endl;
16. }
17. }

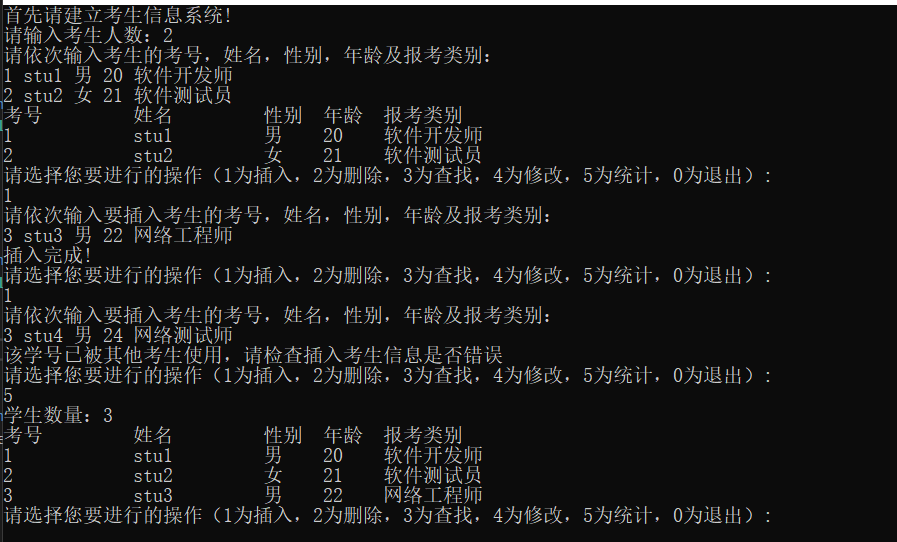
**AVL树接口：**

1. **template**<**typename** Key, **typename** Value>
2. **inline** TreeNode<Key, Value>\* AVLTree<Key, Value>::Insert(Key K, Value V)
3. {
4. **if** (Root == nullptr)
5. {
6. Size++;
7. **return** Root = **new** TreeNode<Key, Value>(K, V);
8. }
9. TreeNode<Key, Value>\* TempCell = Find(K, Root);
10. **if** (TempCell) **return** UncheckChange(TempCell, V);
11. **else**
12. {
13. Size++;
14. Root = Insert(K, V, Root);
15. **return** Root;
16. }
17. }

**AVL树插入内部实现：**

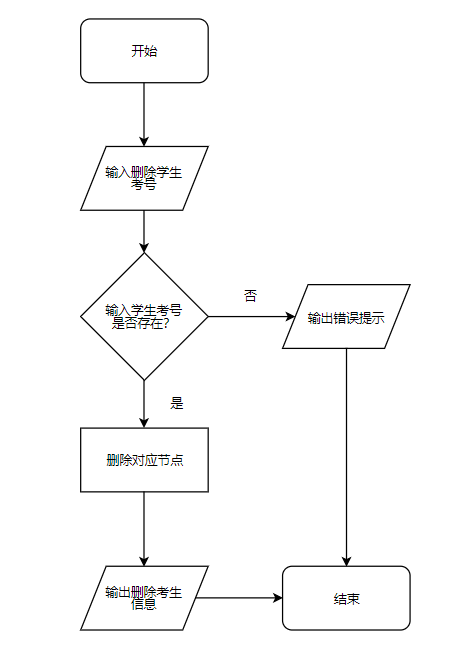
1. **template**<**typename** Key, **typename** Value>
2. **inline** TreeNode<Key, Value>\* AVLTree<Key, Value>::Insert(Key K, Value V, TreeNode<Key, Value>\* P)
3. {
4. **if** (P == nullptr)
5. {
6. P = **new** TreeNode<Key, Value>(K, V);
7. }
8. **else** **if** (K < P->ElementKey)
9. {
10. P->Left = Insert(K, V, P->Left);
11. **if** (GetHeight(P->Left) - GetHeight(P->Right) == 2)
12. {
13. **if** (K < P->Left->ElementKey) P = SingleRotateWithLeft(P);
14. **else** P = DoubleRotateWithLeft(P);
15. }
16. }
17. **else** **if** (K > P->ElementKey)
18. {
19. P->Right = Insert(K, V, P->Right);
20. **if** (GetHeight(P->Right) - GetHeight(P->Left) == 2)
21. {
22. **if** (K > P->Right->ElementKey) P = SingleRotateWithRight(P);
23. **else** P = DoubleRotateWithRight(P);
24. }
25. }
26. P->Height = Max(GetHeight(P->Left), GetHeight(P->Right)) + 1;
27. **return** P;
28. }

### 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

**用户界面接口：**

1. **inline** **void** StudentInformationSystem::Delete()
2. {
3. cout << "请输入要删除考生的考号：" << endl;
4. BigInteger ID;
5. cin >> ID;
6. auto FoundPtr = StudentTree.Find(ID);
7. **if** (FoundPtr == nullptr)
8. {
9. cout << "该学号对应考生不存在，请检查输入考号是否错误" << endl;
10. **return**;
11. }
12. **else**
13. {
14. cout << "你删除的考生的信息为：" << endl;
15. cout << HeadLine << endl;
16. cout << FoundPtr->ElementValue << endl;
17. StudentTree.Delete(ID);
18. cout << "删除完成!" << endl;
19. }
20. }

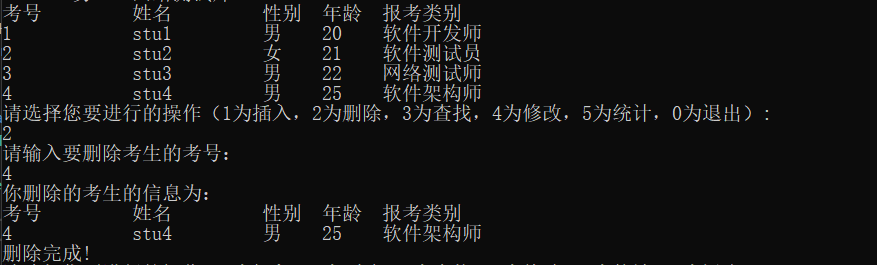
**AVL树接口：**

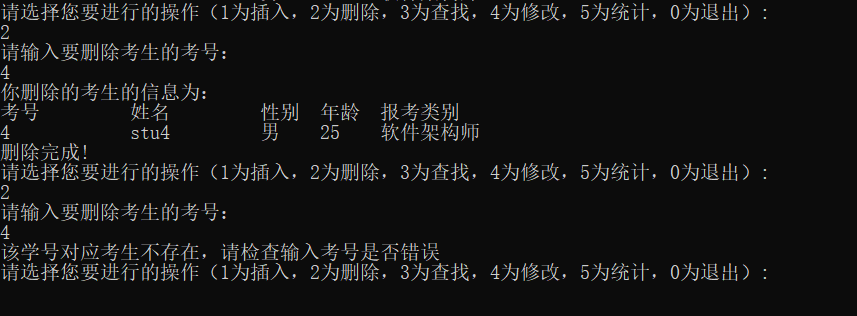
1. **template**<**typename** Key, **typename** Value>
2. **inline** TreeNode<Key, Value>\* AVLTree<Key, Value>::Delete(Key K)
3. {
4. **if** (Root)
5. {
6. **if** (Find(K, Root))
7. {
8. Size--;
9. **return** Root = Delete(K, Root);
10. }
11. **return** nullptr;
12. }
13. **return** nullptr;
14. }

**AVL树删除内部实现：**

1. **template**<**typename** Key, **typename** Value>
2. **inline** TreeNode<Key, Value>\* AVLTree<Key, Value>::Delete(Key K, TreeNode<Key, Value>\* P)
3. {
4. TreeNode<Key, Value>\* TempCell;
5. **if** (P == nullptr) **return** nullptr;
6. **else** **if** (K < P->ElementKey)
7. {
8. P->Left = Delete(K, P->Left);
9. **if** (GetHeight(P->Right) - GetHeight(P->Left) == 2)
10. {
11. TempCell = P->Right;
12. **if** (GetHeight(TempCell->Left) < GetHeight(TempCell->Right)) P = SingleRotateWithRight(P);
13. **else** P = DoubleRotateWithRight(P);
14. }
15. }
16. **else** **if** (K > P->ElementKey)
17. {
18. P->Right = Delete(K, P->Right);
19. **if** (GetHeight(P->Left) - GetHeight(P->Right) == 2)
20. {
21. TempCell = P->Left;
22. **if** (GetHeight(TempCell->Left) > GetHeight(TempCell->Right))P = SingleRotateWithLeft(P);
23. **else** P = DoubleRotateWithLeft(P);
24. }
25. }
26. **else**
27. {
28. **if** (P->Left&&P->Right)
29. {
30. **if** (GetHeight(P->Left) > GetHeight(P->Right))
31. {
32. TempCell = FindMax(P->Left);
33. P->ElementKey = TempCell->ElementKey;
34. P->ElementValue = TempCell->ElementValue;
35. P->Left = Delete(P->ElementKey, P->Left);
36. }
37. **else**
38. {
39. TempCell = FindMin(P->Right);
40. P->ElementKey = TempCell->ElementKey;
41. P->ElementValue = TempCell->ElementValue;
42. P->Right = Delete(P->ElementKey, P->Right);
43. }
44. }
45. **else**
46. {
47. TempCell = P;
48. **if** (P->Left == nullptr) P = P->Right;
49. **else** **if** (P->Right == nullptr)P = P->Left;
50. **delete** TempCell;
51. TempCell = nullptr;
52. }
53. }
54. **if** (P) P->Height = Max(GetHeight(P->Left), GetHeight(P->Right)) + 1;
55. **return** P;
56. }

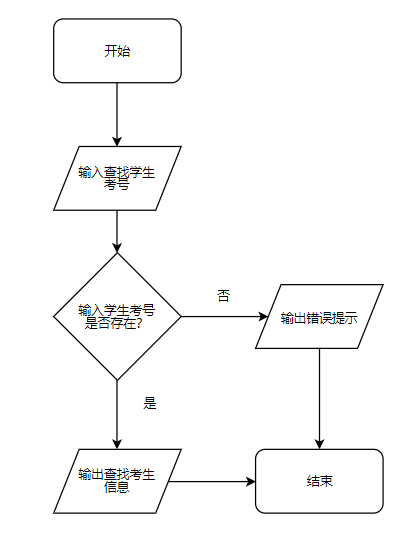
### 3.2.3 删除功能截屏示例





## 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

**用户界面接口：**

1. **inline** **void** StudentInformationSystem::Find()
2. {
3. cout << "请输入要查询考生的考号：" << endl;
4. BigInteger ID;
5. cin >> ID;
6. auto FoundPtr = StudentTree.Find(ID);
7. **if** (FoundPtr == nullptr)
8. {
9. cout << "该学号对应考生不存在，请检查输入考号是否错误" << endl;
10. **return**;
11. }
12. **else**
13. {
14. cout << HeadLine << endl;
15. cout << FoundPtr->ElementValue << endl;
16. }
17. }

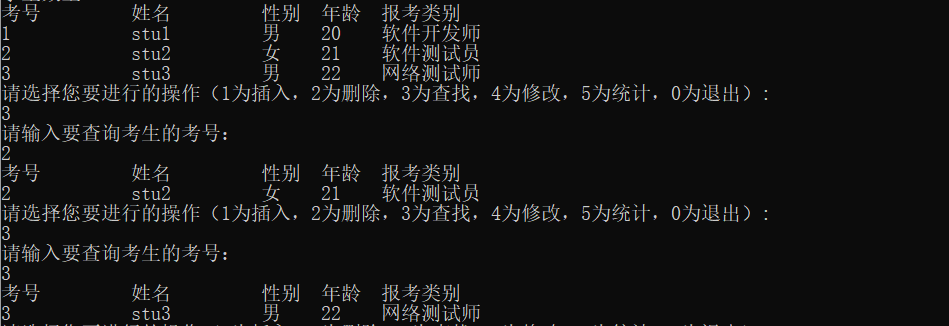
**AVL树接口：**

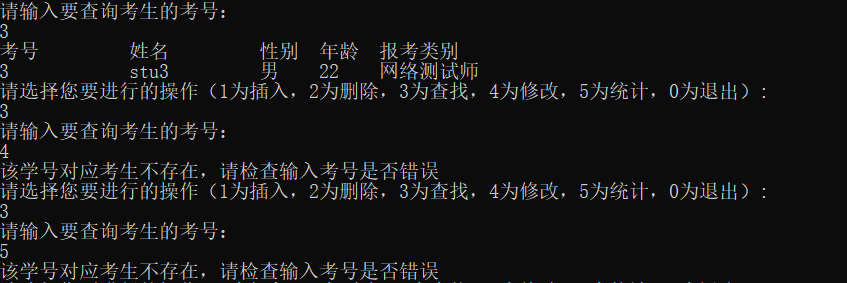
1. **template**<**typename** Key, **typename** Value>
2. **inline** TreeNode<Key, Value>\* AVLTree<Key, Value>::Find(Key K)
3. {
4. **if** (Root == nullptr) **return** nullptr;
5. **return** Find(K, Root);
6. }

**AVL树查找内部实现：**

1. **template**<**typename** Key, **typename** Value>
2. **inline** TreeNode<Key, Value>\* AVLTree<Key, Value>::Find(Key K, TreeNode<Key, Value>\* P)
3. {
4. **if** (P == nullptr) **return** nullptr;
5. **if** (K < P->ElementKey)
6. **return** Find(K, P->Left);
7. **else** **if** (K > P->ElementKey) **return** Find(K, P->Right);
8. **else** **return** P;
9. }

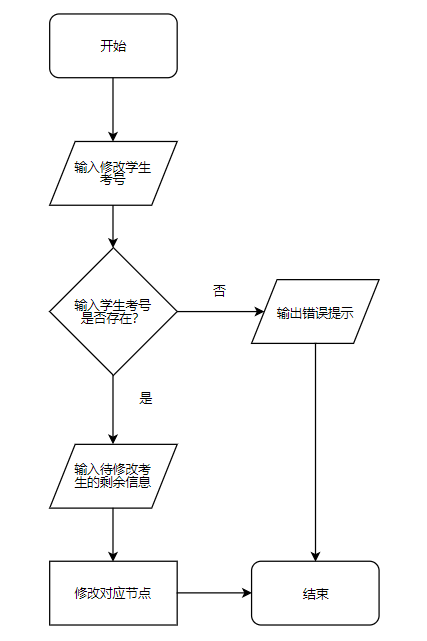
### 3.3.3 查找功能截图示例





## 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

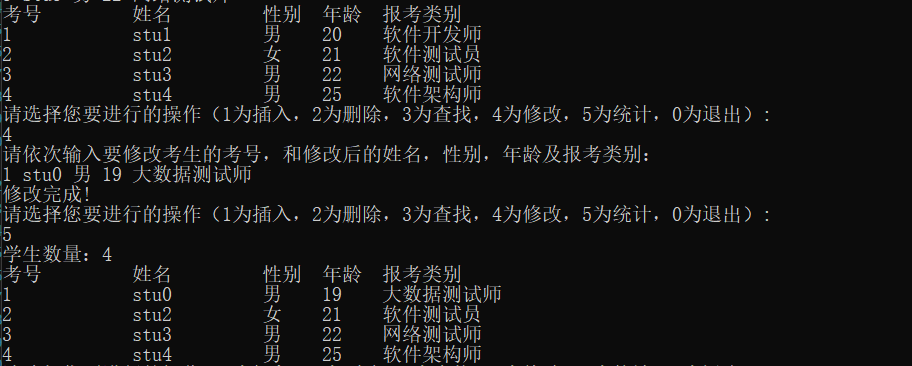
**用户界面接口：**

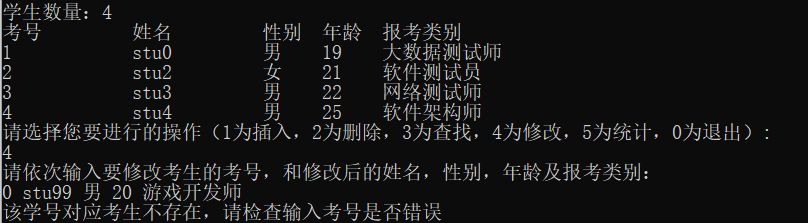
1. **inline** **void** StudentInformationSystem::Change()
2. {
3. cout << "请依次输入要修改考生的考号，和修改后的姓名，性别，年龄及报考类别：" << endl;
4. Student TempStudent;
5. cin >> TempStudent;
6. auto FoundPtr = StudentTree.Find(TempStudent.ID);
7. **if** (FoundPtr == nullptr)
8. {
9. cout << "该学号对应考生不存在，请检查输入考号是否错误" << endl;
10. **return**;
11. }
12. **else**
13. {
14. StudentTree.UncheckChange(FoundPtr, TempStudent);
15. cout << "修改完成!" << endl;
16. }
17. }

**AVL树修改内部实现：**

1. **template**<**typename** Key, **typename** Value>
2. **inline** TreeNode<Key, Value>\* AVLTree<Key, Value>::UncheckChange(TreeNode<Key, Value>\* P, Value V)
3. {
4. P->ElementValue = V;
5. **return** P;
6. }

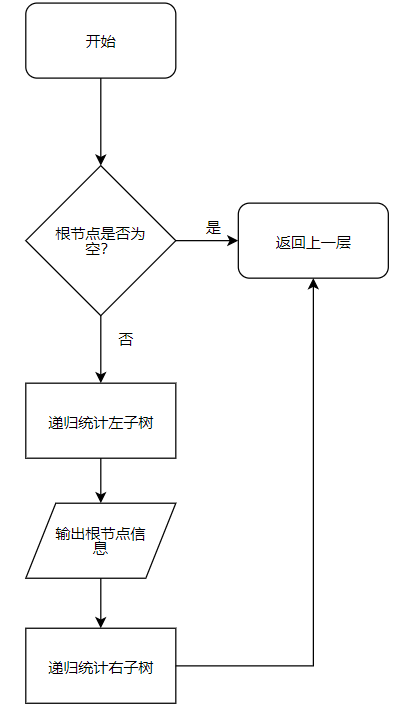
### 3.4.3 修改功能截屏示例





## 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

**用户界面接口：**

1. **inline** **void** StudentInformationSystem::Display()
2. {
3. cout << "学生数量：" << GetStudentNum() << endl;
4. **if** (GetStudentNum() > 0)
5. {
6. cout << HeadLine << endl;
7. StudentTree.DisPlay(cout);
8. }
9. }

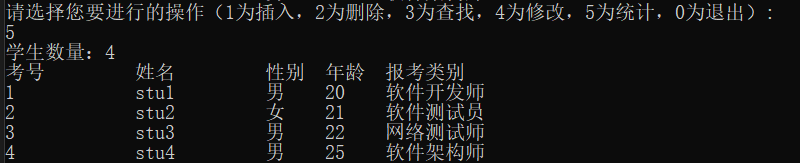
**AVL树接口：**

1. **template**<**typename** Key, **typename** Value>
2. **inline** **void** AVLTree<Key, Value>::DisPlay(ostream & os)
3. {
4. Root->MidVisit(os);
5. }

**AVL树统计内部实现：**

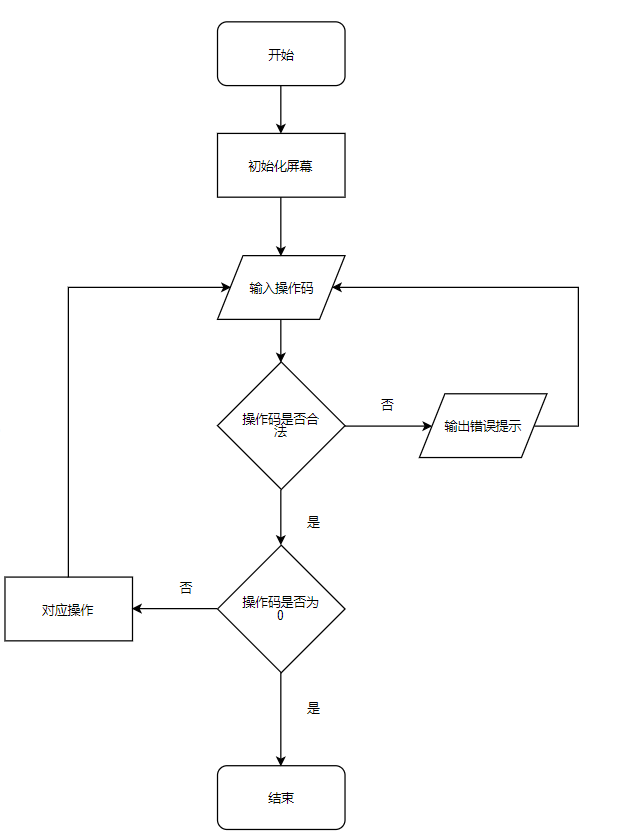
1. **template**<**typename** Key, **typename** Value>
2. **inline** **void** TreeNode<Key, Value>::MidVisit(ostream & os)
3. {
4. **if** (Left) Left->MidVisit(os);
5. os << ElementValue << endl;
6. **if** (Right) Right->MidVisit(os);
7. }

### 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

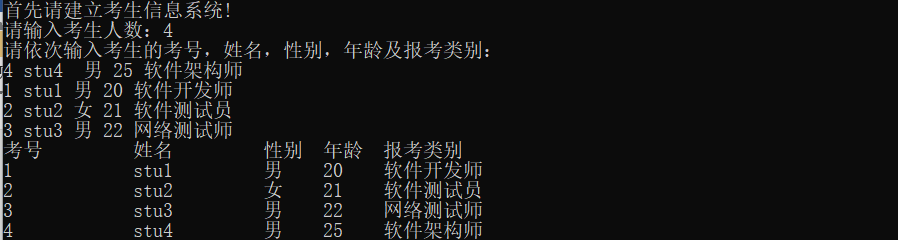


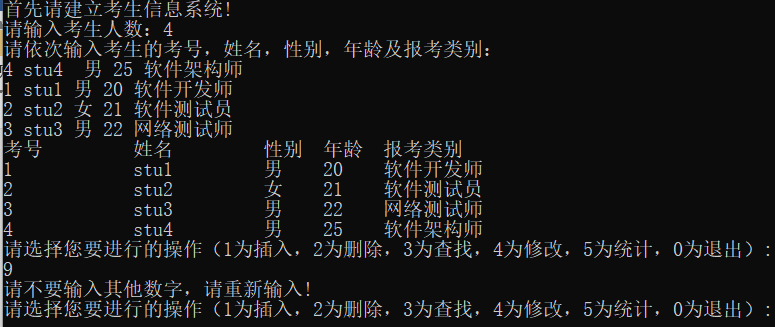
### 3.6.2 总体系统核心代码

1. **int** main()
2. {
3. StudentInformationSystem SIS;
4. SIS.Init();
5. **while** (**true**)
6. {
7. cout << "请选择您要进行的操作（1为插入，2为删除，3为查找，4为修改，5为统计，0为退出）:" << endl;
8. **int** Command;
9. cin >> Command;
10. **switch** (Command)
11. {
12. **case** 1:
13. SIS.Insert();
14. **break**;
15. **case** 2:
16. SIS.Delete();
17. **break**;
18. **case** 3:
19. SIS.Find();
20. **break**;
21. **case** 4:
22. SIS.Change();
23. **break**;
24. **case** 5:
25. SIS.Display();
26. **break**;
27. **case** 0:
28. cout << "感谢您的使用！" << endl;
29. **return** 0;
30. **default**:
31. cout << "请不要输入其他数字，请重新输入!" << endl;
32. **break**;
33. }
34. }
35. **return** 0;
36. }

### 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**：stu3 男 22 网络工程师

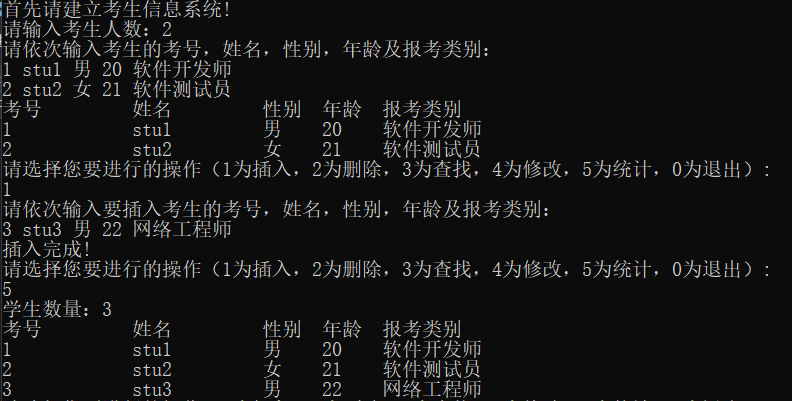
**预期结果**：

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

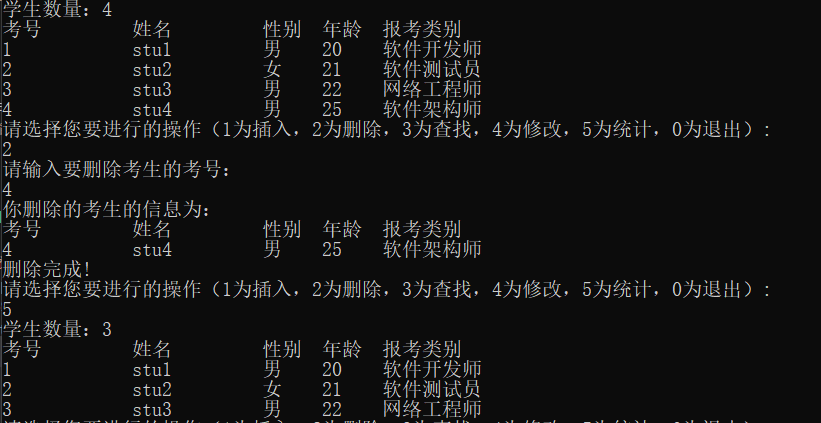
**预期结果：**

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：**



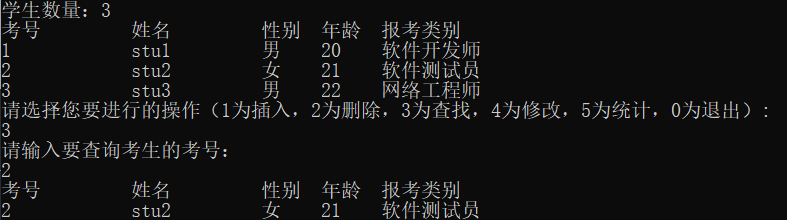
### 4.1.3 查找功能测试

**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：**



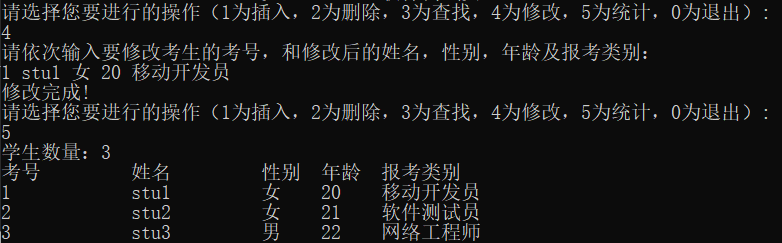
### 4.1.4 修改功能测试

**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

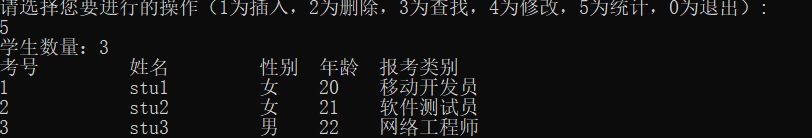
**预期结果：**

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：**



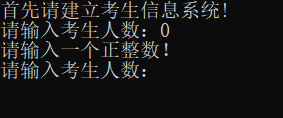
## 4.2 边界测试

### 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：**

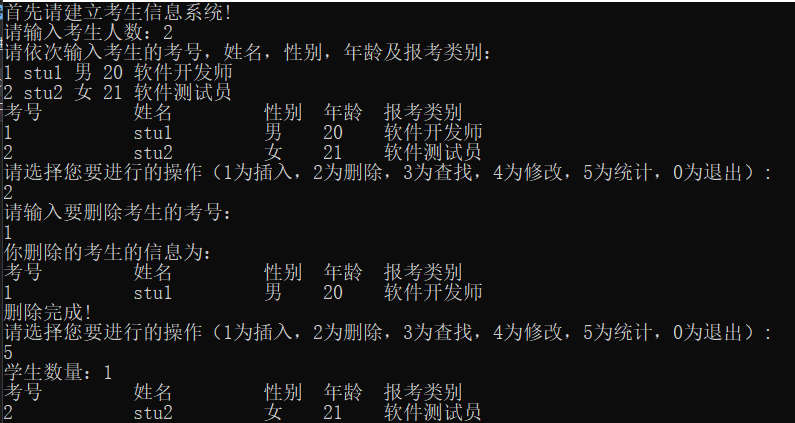


### 4.2.2 删除根节点

**测试用例：**删除根结点

**预期结果：**程序正常运行，不崩溃。

**实验结果：**

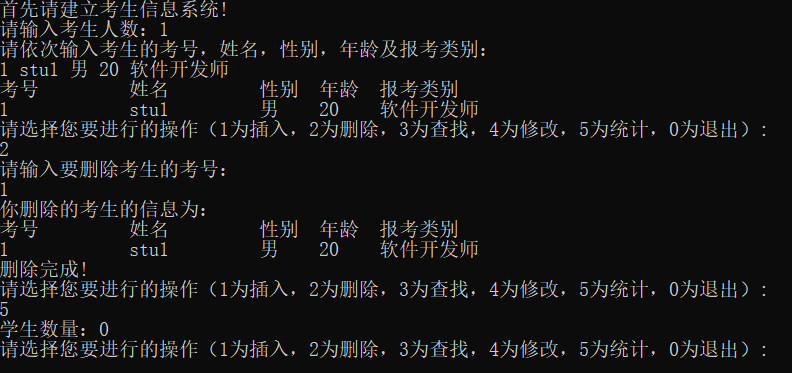


### 4.2.3 删除后AVL树为空

**测试用例：**删除前AVL树内只有一个结点，删除后AVL树为空

**预期结果：**程序正常运行，不崩溃。

**实验结果：**

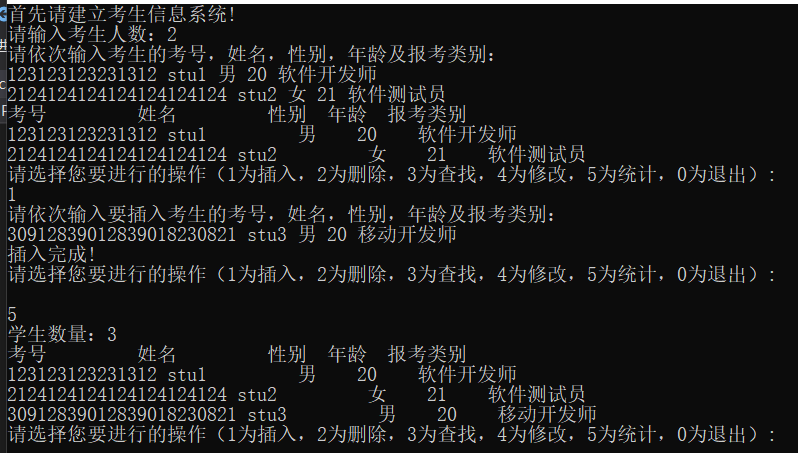


### 4.2.4 插入考生考号极大

**测试用例：**插入的考生考号考号极大

**预期结果：**程序正常运行，考生按考号大小正确排序

**实验结果：**

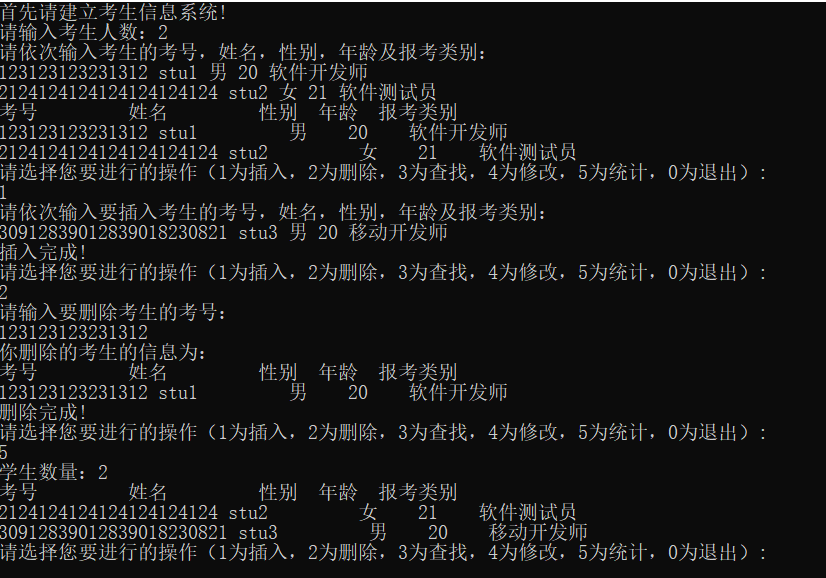


### 4.2.5 删除考生考号极大

**测试用例：**删除的考生考号考号极大

**预期结果：**程序正常运行，考生按考号大小正确排序

**实验结果：**

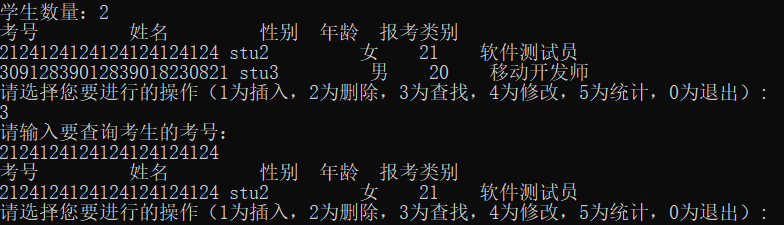


### 4.2.6 查找考生考号极大

**测试用例：**查找的考生考号考号极大

**预期结果：**程序正常运行，正确显示结果

**实验结果：**

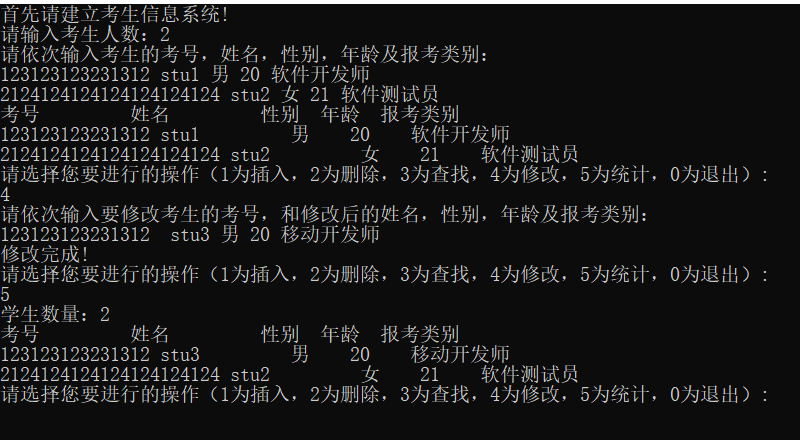


### 4.2.7 修改考生考号极大

**测试用例：**修改的考生考号考号极大

**预期结果：**程序正常运行，正确显示结果

**实验结果：**



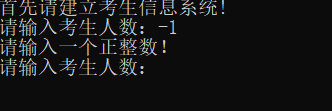
## 4.3 出错测试

### 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：**

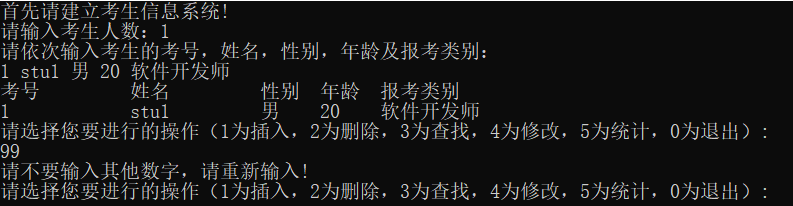


### 4.3.2 操作码错误

**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：**

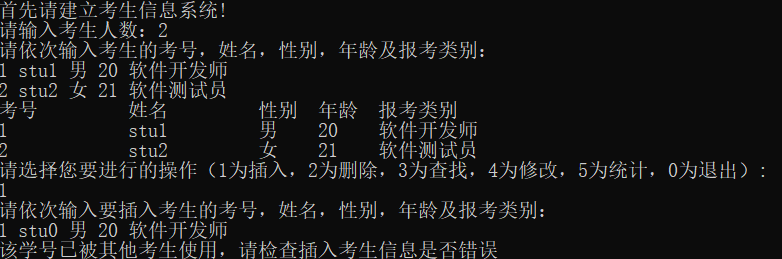


### 4.3.3 插入考生的考号已经存在

**测试用例：**AVL树里有两条记录，向AVL树里插入已经存在的考号

**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：**

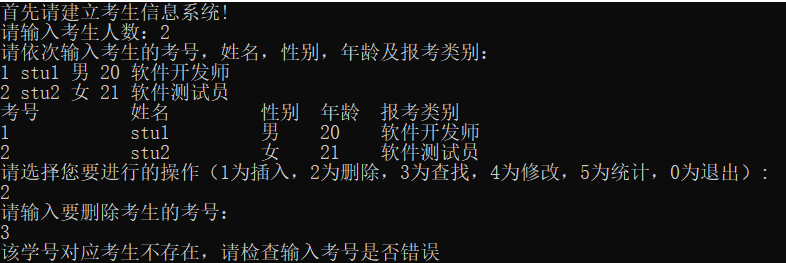


### 4.3.4 删除考号不存在

**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：**

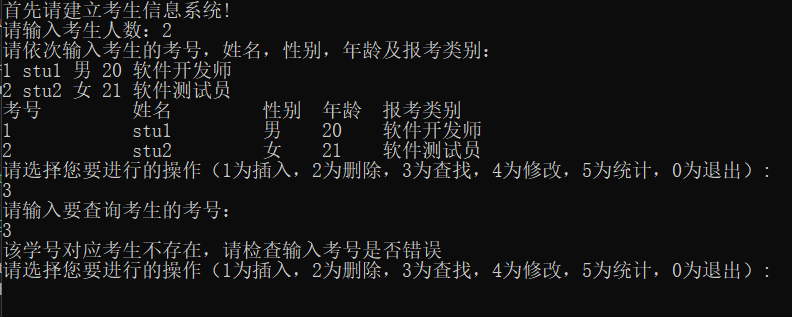


### 4.3.5 查找考号不存在

**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：**

