数据结构课程设计

项目说明文档

考试报名系统

作者姓名: 汪明杰

学　　号: 1851055

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

1.1 项目背景

对于每一所学校而言，都有着举行考试的需求。因此，考试报名系统也就成为了一个学校所不可或缺的一部分。同时，也正因为它对于学校的教务处的重要作用，一个良好的考试报名系统应该能为用户提供充分的功能。

而事实上，由于数量逐渐增加的考试人数，考试报名工作给当前的高校报名工作带来了新的挑战，也给教务管理部门增加了很大的工作量。例如在我们日常生活中，比较常接触到的考试就有四、六级考试、普通话考试、计算机水平考试等等，这些考试报名者都需要能够合理的管理，从而实现分配考场等功能。随着学生数量和考试数量的日益庞大，如何管理如此庞大的数据显得极为复杂，传统的手工管理工作量大且容易出错。

随着计算机科学技术的不断成熟，使用计算机对考试报名系统进行管理，具有手工管理所无法比拟的优势。这些优点能够极大地提高学校和学生的效率，也是学校走向信息化、科学化、国际化的重要条件。因此，开发一套考试报名系统具有十分重要的意义。

1.2 项目需求分析

一个完善的考试报名系统应当满足以下需求：

- ✓ **功能完善**

  作为一个最基本的数据库，增删改查是最基本的功能。除了这些基本功能外，还应该有人数统计，按类别和年龄筛选，信息有效性检查的功能。

- ✓ **可复用**

  为了避免下次重新导入信息，本系统应当支持存储信息的功能。

  同时，也应该为使用者提供保存信息文件，以便下次重复使用的功能。

- ✓ **执行效率高**

  能够存储的数据条数应该足够多，至少能够存储一个学校（万条以上）学生信息。

  即使在存入大量数据后进行查询也应该有较快的响应速度，其内部使用的数据结构和算法应该具有较低的时间和空间复杂度。

- ✓ **健壮性**

  对于错误的输入如输入了已存在的考号等，程序应该做到不崩溃并且给予一定的提示。

- ✓ **美观的界面**

  在第一次运行时，程序会提示用户初始化考生信息系统。

  在打印考生信息时，应该注意页面篇幅，若信息条数极多，则应该只打印前 50 条。

- ✓ **数据可视化**

  在进行数据统计的时候，应当尽可能将数据信息通过可以直观感知的信息进行体现。

1.3 项目要求

1.3.1 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。

1.3.2 输入格式

输入需要执行的操作，以及考生的信息（包括准考证号、姓名、年龄和报考类别）

1.3.3 输出格式

输出查找学生的信息

1.3.4 项目示例

2 项目设计

2.1 数据结构设计

本系统是一个存储了学生类的存储系统，关键操作是大量对于数据的增加、删除、修改、查找等。因此，选择的数据结构要应该能够快速的对数据执行以上操作。本系统采用了链表作为底层数据。

2.2 类设计

经典的链表一般包括两个抽象数据类型（ADT）——链表结点类（ListNode）与链表类（List），而两个类之间的耦合关系可以采用嵌套、继承等多种关系。

为了实现代码的复用性，本系统实现了一个**链表**。采用 struct 描述链表结点类（ListNode），这样使得链表结点类（List）可以直接访问链表结点而不需要定义友元关系。本系统实现的链表结构各种操作的时间复杂度如下：

- ✓ 插入操作：O(n)
- ✓ 删除操作：O(n)
- ✓ 查询操作：O(n)
- ✓ 遍历操作：O(n)

2.2.1 结点类（ListNode）

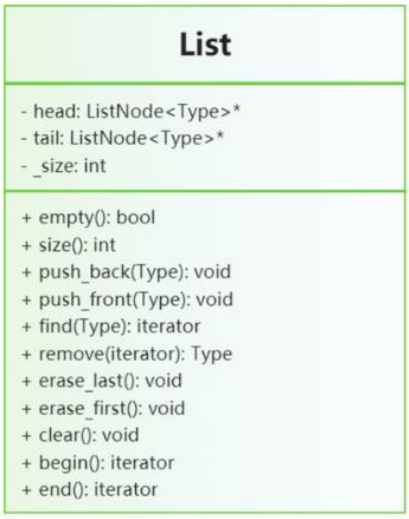
链表的结点中存储的数据有链表数据、后继结点和前驱结点。其 UML 图如下所示：

2.2.2 双向链表类（List）

链表的实现原理大同小异，不同之处在于：是否带头结点、是否带尾结点、每一个结点是否带前驱结点等。为了使得链表中各种操作的时间复杂度都尽可能

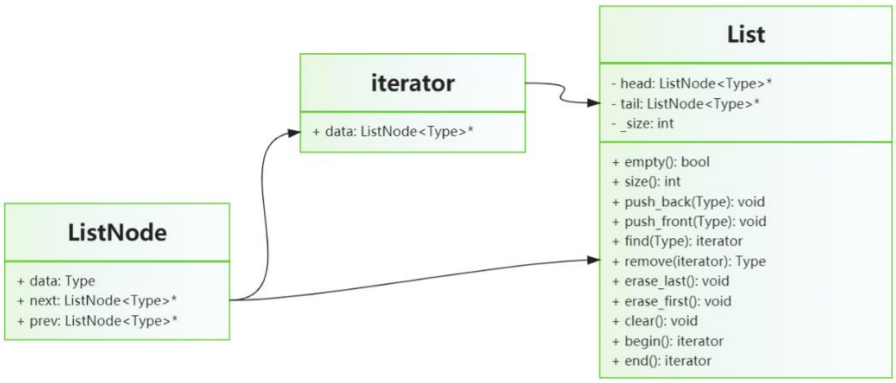
低，因此这里选择了带头结点和尾结点的双向链表来实现链表中的各种操作。也即：选择了牺牲空间来达到降低时间复杂度的效果。

链表的 UML 图如下所示：



为了便于对链表进行遍历、插入、删除、查找等操作，增加了一个 iterator 类。iterator 类内部存储一个链表节点指针，同时，通过运算符重载相应的自增、自减、判等等操作。

iterator 类、ListNode 类和 List 类的关系如下图所示：



其中，List 类中的主要函数如下所示：

◆ `inline int size()const`

返回链表中结点的个数，不包括头结点。

◆ `inline bool empty()const`

判断链表是否为空，也即链表中结点的个数是否为 0。

◆ `void push_back(Type data)`

在链表尾部插入新的数据，也即新增一个结点并且加入到链表末端。

◆ `void push_front(Type data)`

在链表头部插入新的数据，也即新增一个结点并且加入到链表的头部。

◆ `iterator find(const Type& data)const`

在链表中查找值为 data 的元素是否存在，返回该位置的迭代器，若查找失败返回空指针对应的迭代器。

◆ `Type remove(iterator index)`

移除迭代器所处位置的元素，返回移除位置元素的值。

◆ `void erase_last()`

移除链表末端的元素，即最后的结点。

◆ `void erase_first()`

移除链表首端的元素，即第一个结点。

◆ `void clear()`

清空链表，即删除链表中所有的结点。

◆ `iterator begin()`

返回链表第一个元素所在位置的迭代器。

◆ `iterator end()`

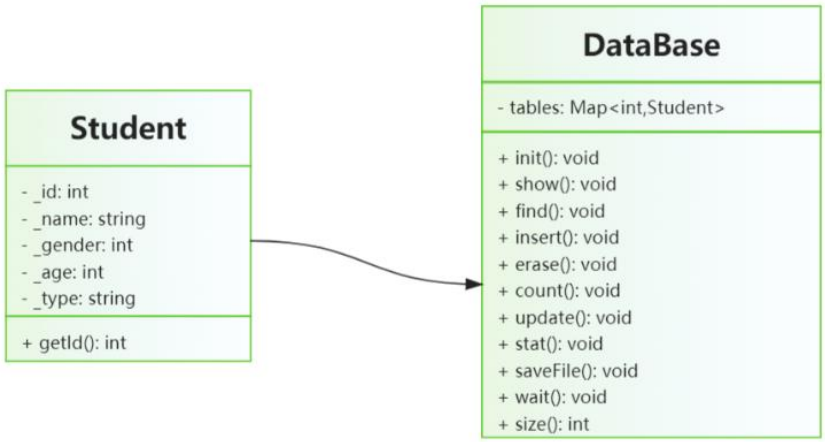
返回链表尾结点后的空结点的迭代器。

2.2.3 数据库类（DataBase）

针对于考试报名系统，其内部存储的数据应当是每一个考生。每一个考生都有着相应的姓名、性别、年龄、报考类别，因此首先应当定义一个学生类（Student）来存储学生信息。

此外，为了便于管理学生的信息，定义一个数据库类（DataBase）。内部使用 Map 存储学生的数据信息，同时提供各种操作方法，如初始化、添加考生、删除考生、修改考生信息、统计考生信息等。

基于上述原理，这两个类的 UML 图及关系如下图所示：



数据库类的关键操作函数及其介绍如下所示：

◆ `void init()`

初始化数据库，本系统中可以通过手动输入和读取文件两种方法进行初始化内部的学生信息。

◆ `void show()`

打印学生信息。

◆ `void insert()`

向数据库中插入一条新的学生数据。

◆ `void erase()`

删除数据库中指定的学生数据。

◆ `void find()`

在数据库中根据学号查询指定的学生信息。

◆ `void count()`

打印数据库系统内的考生总数。

◆ `void update()`

更新考生信息。

◆ `void stat()`

统计数据库内考生信息，包括男女比例、年龄比例等。

◆ `void saveFile()`

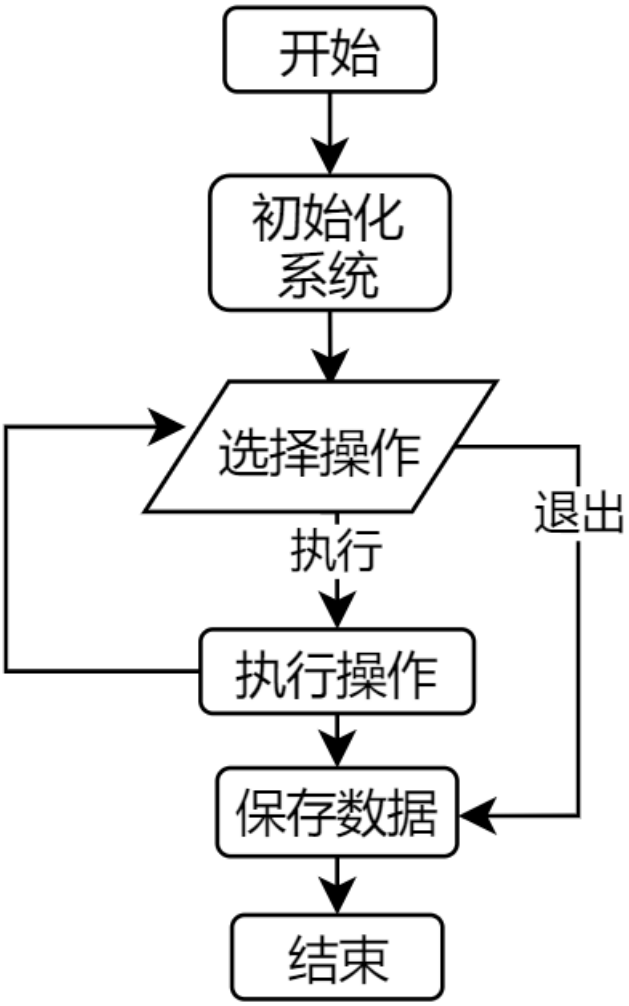
将数据库保存到本地文件中，以便下次打开系统操作数据。

通过以上的类以及数据库类，并且加以耦合，即完成了考试报名系统。

3 项目实现

3.1 项目主体功能

3.1.1 项目主体功能流程图



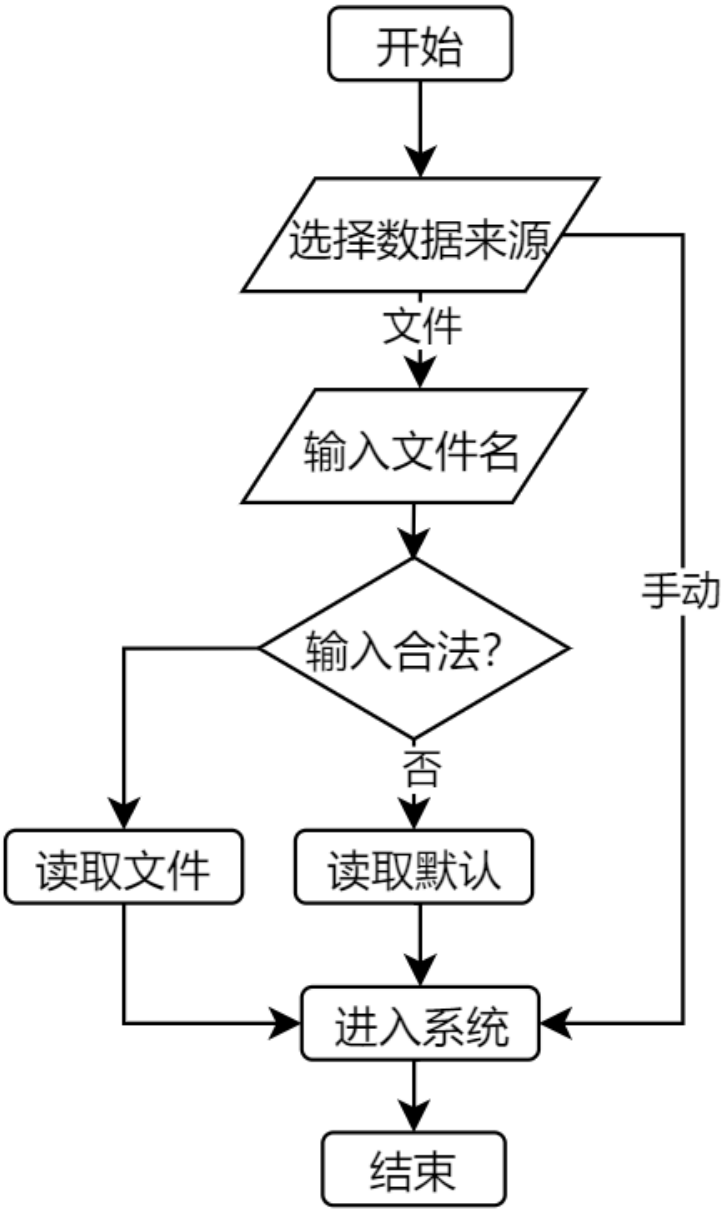
3.1.2 项目主体功能代码

```
int main()
{
    DataBase db;
    db.init();
```

```
    string operation; bool quit = false;
    while (!quit)
    {
        cout << "请选择操作（1 为插入，2 为删除，3 为查找，4 为修改，5 为"
            "统计，6 为保存数据，0 为退出系统）\n\n";
        cin >> operation;
        switch (operation[0])
        {
        case '0':
            quit = true;
            cout << "退出系统前是否需要保存数据?" << endl;
            cout << "1.是" << endl;
            cout << "2.否(默认)" << endl;
            int index;
            cin >> index;
            if (cin.fail() || index != 1)
                return 0;
            else db.saveFile();
            break;
        case '1':
            db.insert();break;
        case '2':
            db.erase();break;
        case '3':
            db.find();break;
        case '4':
            db.update();break;
        case '5':
            db.stat();break;
        case '6':
            db.saveFile();break;
        default:
            cout << "未知操作,请重新输入" << endl;
            cout << "(输入提示: "
                "1 为插入，2 为删除，3 为查找，4 为修改，5 为统计，0 为"
                "退出系统)"
                << endl; break;
        }
        if (db.size() < 50) db.show();
        cout << "当前系统内共有 " << db.size() << " 条数据" << endl
            << endl;
    }
    return 0;
}
```

3.2 项目初始化

3.2.1 项目初始化流程图



3.2.2 项目初始化代码

本系统为用户提供了两种初始化的方法：

- ✓ 读取文件
- ✓ 手动输入

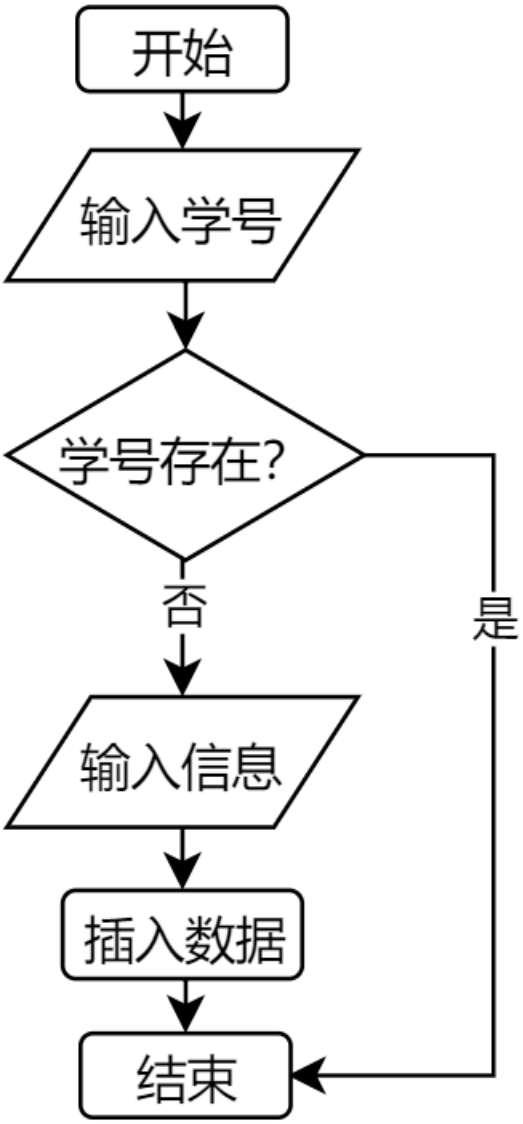
用户可以选择通过读取文件的方法初始化数据库内的数据，也可以选择新建一个数据库后手动键入数据。

```
void DataBase::init()
{
    int count_Student = 0;
    cout << "首先请建立考生信息系统! " << endl;
    cout << "请选择初始化数据库方法:" << endl;
    cout << "1.从文件中读取(默认)" << endl;
    cout << "2.手动输入" << endl;
    int temp;
    cin >> temp;
    switch (temp)
    {
    case 1:
    {
        cout << "请输入要存储文件的名称(输入 0 代表读取默认文件)" << endl;
        string loadFile;
        cin >> loadFile;
        if (loadFile.length() == 0)
        {
            cout << "无效名称，自动读取名为 Student.txt 的文件" << endl;
            loadFile = "Student.txt";
        }
        else if (loadFile == "0")
        {
            cout << "开始读取默认文件" << endl;
            loadFile = "Student.txt";
        }
        else
        {
            loadFile += ".txt";
        }
        ifstream file(loadFile);
        cout << endl << "开始读取数据:" << endl;

        Student temp;
```

```
        while (file.peek() != EOF)
        {
            file >> temp;
            this->insertProperPlace(temp);
        }
        cout << "成功读取数据！" << endl;
        this->show();
        break;
    }
    case 2:
    {
        cout << "请输入考生人数：" << endl;
        //循环输入，直到输入合理正整数
        while (true)
        {
            cin >> count_Student;
            if (cin.fail() || count_Student < 0)
            {
                cout << "请输入一个正整数！" << endl;
                cin.clear();
                cin.ignore(1024, '\n');
            }
            else break;
        }
        if (count_Student == 0)
            break;
        cout << "请依次输入考生的考号，姓名，性别，年龄及报考类别！" << endl;
        Student temp;
        for (int i = 0; i < count_Student; ++i)
        {
            cin >> temp;
            if (table.find(temp) != table.end())
            {
                cout << "该考号对应的考生已存在！" << endl;
            }
            else
            {
                this->insertProperPlace(temp);
            }
        }
        this->show();
        break;
    }}
}
```

3.3 插入数据

3.3.1 插入数据流程图



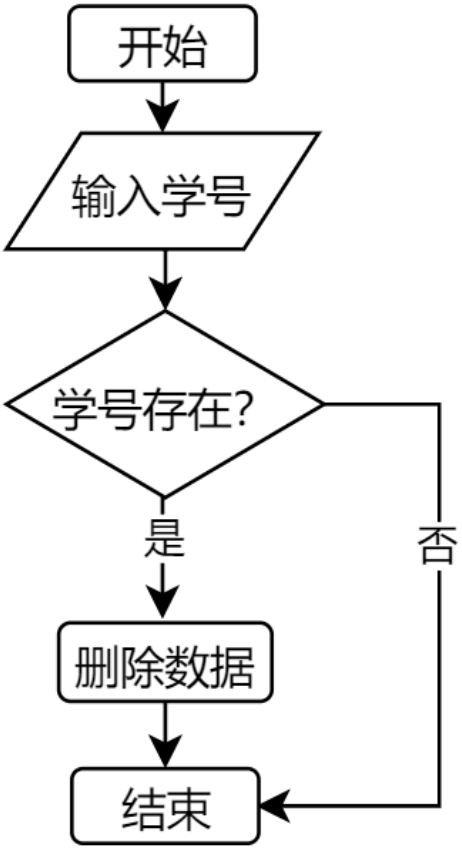
3.3.2 插入数据代码

```
void DataBase::insert()
{
    cout << "请输入要插入的考生的考号：";
    int StudentID;

    while (true)
```

```
    {
        cin >> StudentID;
        if (cin.fail())
        {
            cout << "请输入一个数字" << endl;
            cin.clear();
            cin.ignore(1024, '\n');
        }
        else
            break;
    }

    Student temp;
    temp._id = StudentID;
    //查找
    if (table.find(temp) != table.end())
    {
        cout << "该考号对应的考生已存在! " << endl;
    }
    else
    {
        //插入数据
        Student temp(StudentID);
        cout << "请输入该考生的姓名，性别，年龄及报考类别! ";
        input_no_id(cin, temp);
        this->insertProperPlace(temp);
    }
}
```

3.4 删除数据

3.4.1 删除数据流程图

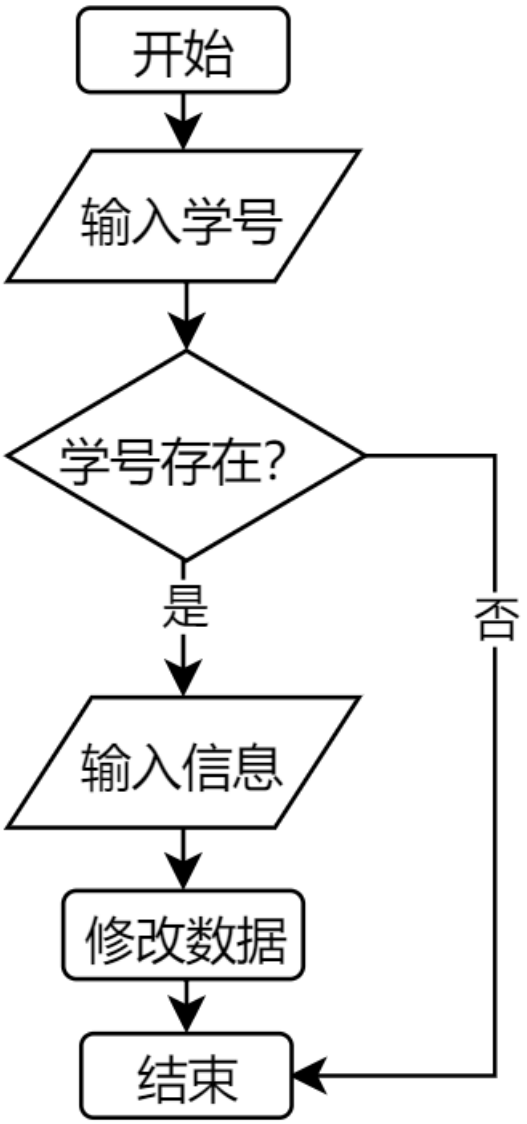


3.4.2 删除数据代码

```
void DataBase::erase()
{
    int id;
    cout << "请选择您要删除的考生的考号: ";
    cin >> id;auto it = table.find(id);  //迭代器
    if (it == table.end())
    {
        cout << "找不到考号为 " << id << " 的考生" << endl;
        return; }
    cout << "您删除的考生信息是: " << (*it)._id << endl;
    table.remove(it);  //使用迭代器删除
}
```

3.5 修改数据

3.5.1 修改数据流程图

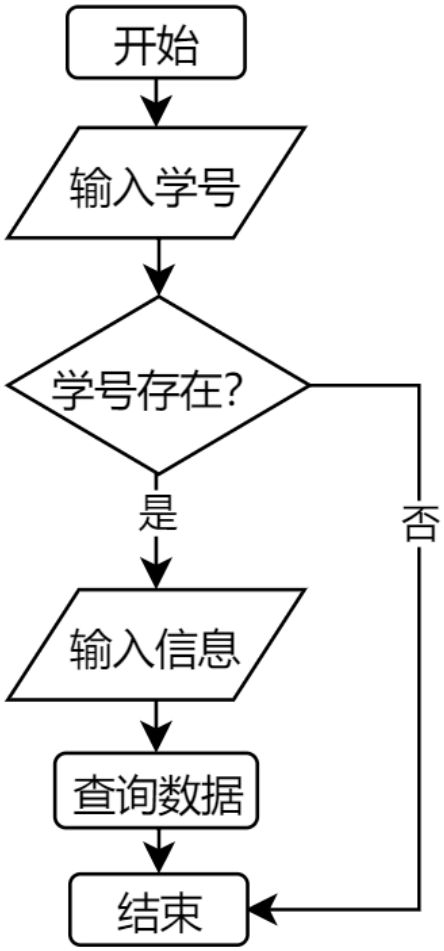


3.5.2 修改数据代码

```
void DataBase::update()
{
    int id;
    cout << "请选择您要修改的考生的考号: ";
    cin >> id;
    auto it = table.find(id);  //迭代器
    if (it == table.end())
    {
        cout << "找不到考号为 " << id << " 的考生" << endl;
        return;
    }
    cout << "请依次输入要修改的考生的姓名，性别，年龄及报考类别! " << endl;
    Student stu(id);
    input_no_id(cin, stu);
    //更新
    *it = stu;
}
```

3.6 查询数据

3.6.1 查询数据流程图



3.6.2 查询数据代码

```
void DataBase::find()
{
    cout << "请输入要查找的考生的考号：";
    int StudentID;
    cin >> StudentID;
    auto result = table.find(StudentID);
    if (result != table.end()) cout << (*result);
    else cout << "不存在考号为" << StudentID << "的考生！" << endl;
```

```
    wait();
}
```

3.7 统计数据

项目提供了函数用于统计系统内部的数据，包括考生总数、男女比例等信息。代码如下所示：

```
void DataBase::stat()
{
    int sumStudent = table.size();
    int sumMale = 0;
    for (auto i = table.begin(); i != table.end(); ++i)
    {
        if ((*i)._gender == Student::male)
            ++sumMale;
    }

    Vector<Pair<int, int>> age;
    int sumAge = 0, index = -1;
    int sumTeenager = 0;


    if (sumStudent == 0)
    {
        cout << "当前系统内暂无考生! " << endl;
        return;
    }
    cout << endl;
    cout << "----------------------------------------" << endl;
    cout << "考生总数:" << sumStudent << endl;
    cout << "----------------------------------------" << endl;
    cout << "性别统计" << endl;

    int drawSumMale = sumMale, drawSumFemale = sumStudent - sumMale;
    //调整个数，让绘图更美观
    while (drawSumMale > 10 && drawSumFemale > 10)
    {
        drawSumMale /= 10;
        drawSumFemale /= 10;
    }
    while (drawSumMale > 16)
        drawSumMale /= 2;
    while (drawSumFemale > 20)
```

```
        drawSumFemale /= 2;


    cout << "男生总数:";
    for (int i = 0; i < drawSumMale; ++i)
        cout << "■";

    cout << sumMale << endl;
    cout << "女生总数:";
    for (int i = 0; i < drawSumFemale; ++i)
        cout << "■";
    cout << sumStudent - sumMale << endl;

    wait();
}
```

4 项目测试

4.1 项目初始化测试

4.1.1 读取文件测试

测试用例：1 0

预期结果：从本地文件中读取初始化的数据库

实验结果：

4.1.2 手动输入测试

测试用例：

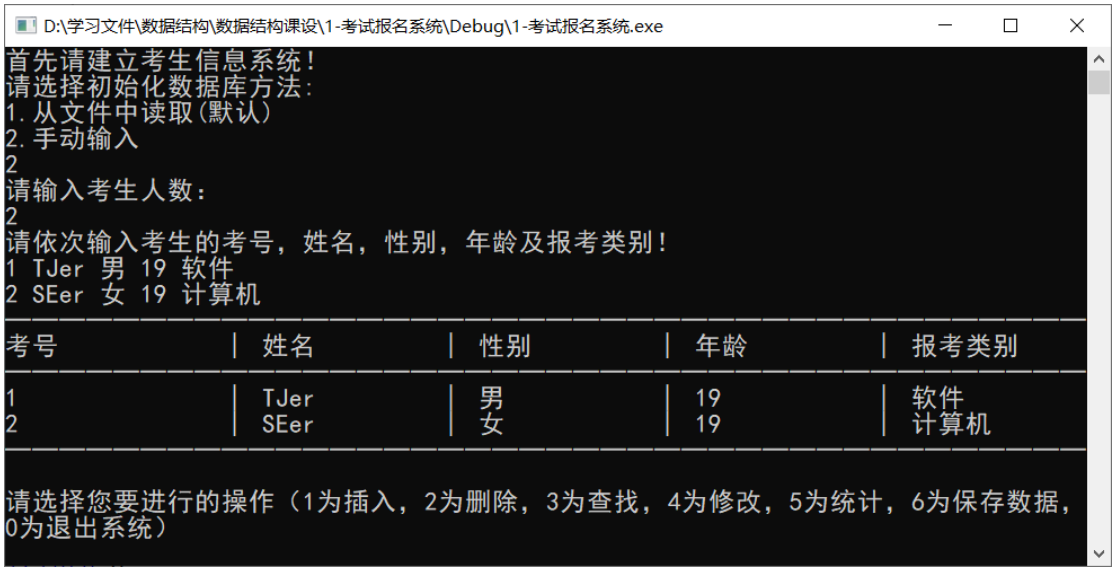
2 2

1 TJer 男 19 软件

2 SEer 女 19 计算机

预期结果：空数据库

实验结果：



4.2 插入测试

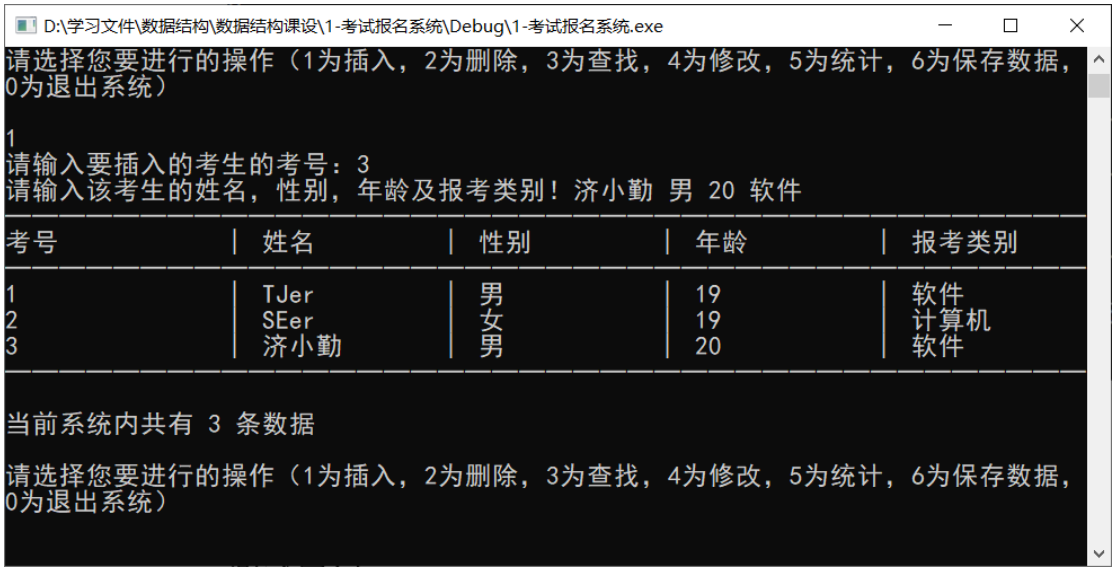
测试用例：

1

3 济小勤 男 20 软件

预期结果：成功插入新数据后的数据库

实验结果：

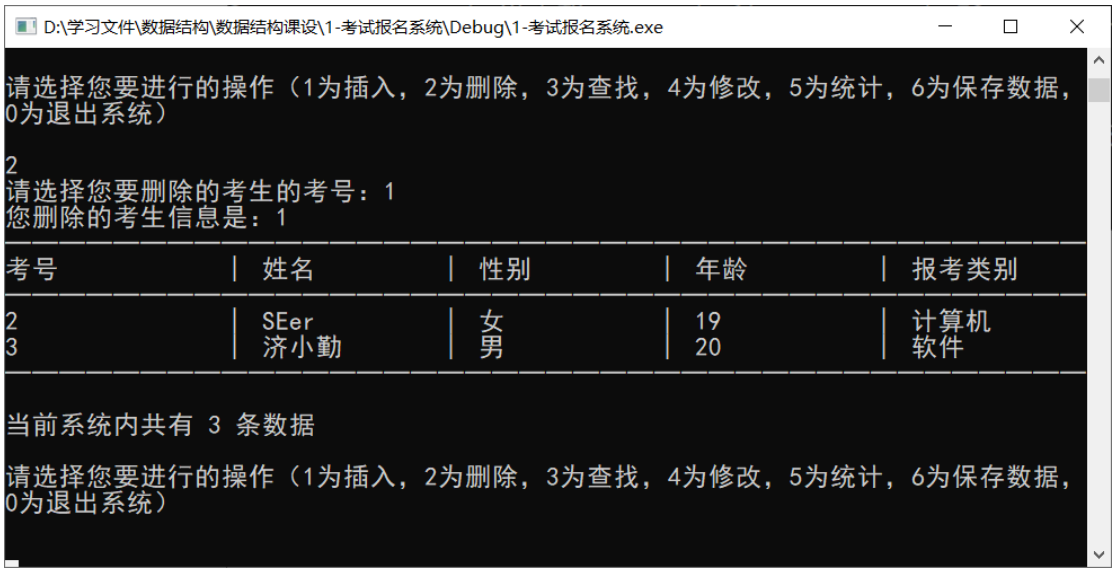


4.3 删除测试

测试用例：2 1

预期结果：删除学号为 1 的考生后的数据库

实验结果：

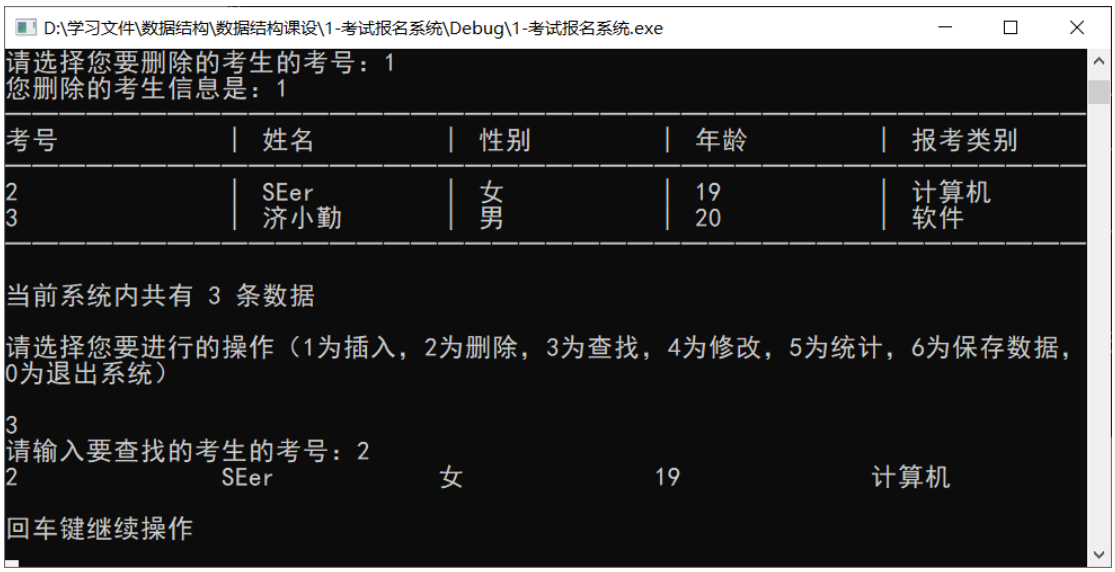


4.4 查找测试

测试用例：3 2

预期结果：学号为 2 的考生的信息

实验结果：



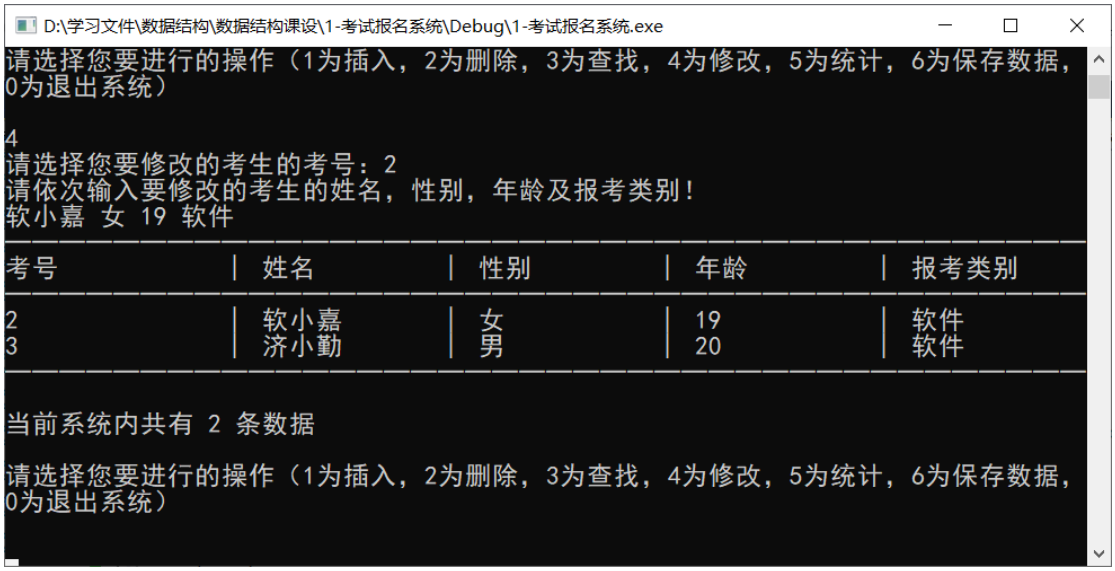
4.5 修改测试

测试用例：4 2

软小嘉 女 19 软件

预期结果：考号为 2 的考生信息被修改后的数据库

实验结果：



4.6 统计测试

测试用例：5

预期结果：统计当前数据库的信息

实验结果：

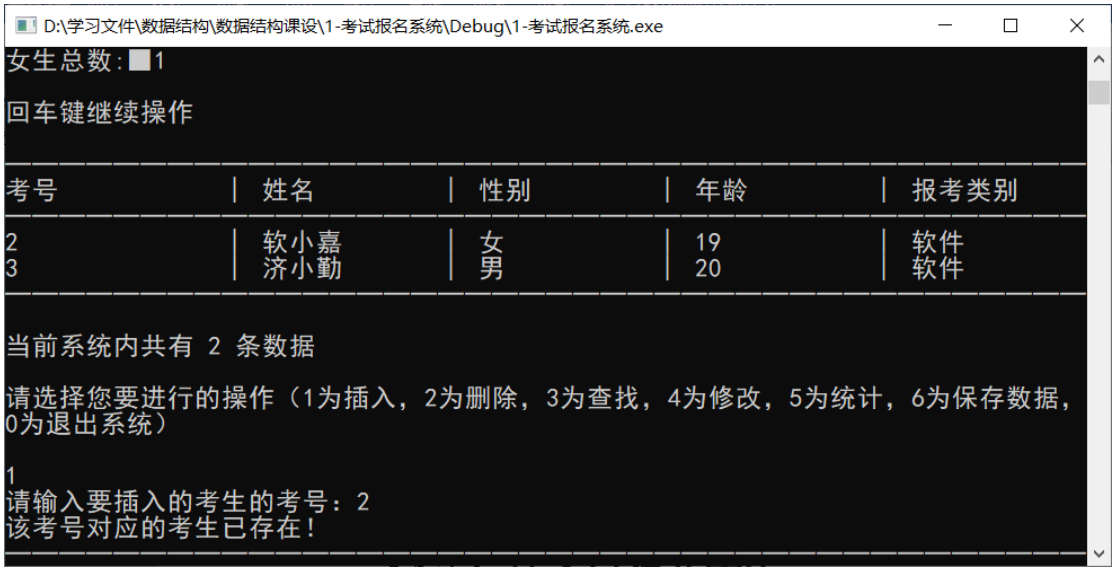
4.7 边界测试

4.7.1 插入已存在考生

测试用例：1 2

预期结果：提示无法插入

实验结果：

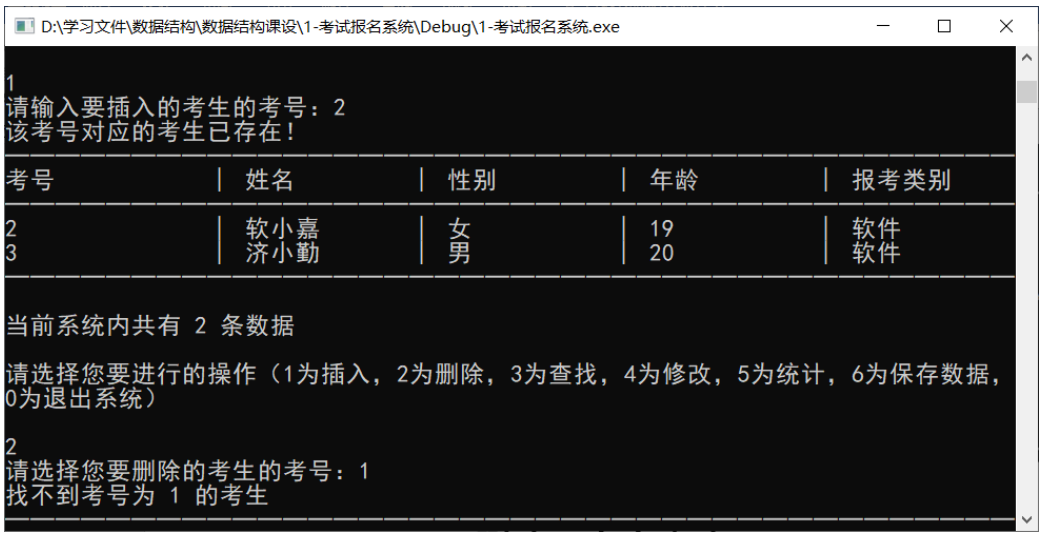


4.7.2 删除不存在考生

测试用例：2 1

预期结果：提示删除失败

实验结果：



4.7.3 查找不存在考生

测试用例：3 1

预期结果：提示查找失败

实验结果：

4.7.4 修改不存在考生

测试用例：4 1

预期结果：提示修改失败

实验结果：