数据结构课程设计项目说明文档

——考试报名系统

作 者 姓 名： 马威

学 号： 2151294

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

****

目 录

[1 项目分析 1](#_Toc495668153)

[1.1 项目背景 1](#_Toc495668154)

[1.2 项目要求 1](#_Toc495668154)

[1.3 项目需求分析 2](#_Toc495668155)

[2 项目设计 3](#_Toc495668156)

[2.1 数据结构设计 3](#_Toc495668157)

[2.2 类设计 3](#_Toc495668158)

2.2.1 家族成员类（FamilyMem） 3

2.2.2 家族树类（FamilyTree） 3

2.2.3 向量类（Vector） 5

2.2.4 字符串类（String） 7

2.3 算法设计 9

2.3.1 算法思路 9

2.3.2 性能评估 10

2.3.3 一般操作过程 10

·流程图表示 10

·完善家谱代码实现 11

·添加家庭成员代码实现 12

·解散局部家庭代码实现 13

·更改家庭成员姓名代码实现 13

·打印全家福代码实现 14

·查找家庭成员代码实现 14

·展示局部家庭代码实现 15

2.3.4 项目主体部分 16

·流程图表示 16

·代码实现 17

[3 项目测试 1](#_Toc495668161)9

[3.1 建立家谱 1](#_Toc495668162)9

[3.2 完善家谱 1](#_Toc495668166)9

[3.3 完成多个家庭建立 2](#_Toc495668170)0

[3.4 添加家庭成员 2](#_Toc495668174)0

[3.5 解散局部家庭 2](#_Toc495668178)1

[3.6 更改家庭成员姓名 2](#_Toc495668182)1

[3.7 查找家庭成员（找到一个） 21](#_Toc495668182)

3.8 查找家庭成员（找到多个） 22

3.9 展示局部家庭 22

3.10 保存族谱 23

3.11 从多个查找结果中选择一个的情况 23

3.12 退出程序 24

3.13 多代单传的情况 24

3.14 一代多子女的情况 24

**1.项目分析**

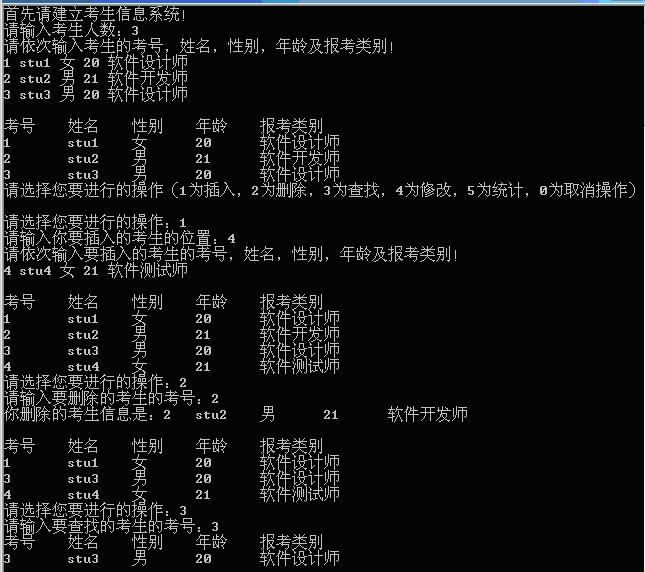
1.1 项目背景

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。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

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）

项目示例：



1.3 项目需求分析

对于实现考试报名系统的程序，需考虑以下需求：

**·正确性**

程序应当能够按照要求对建立考生系统、对学生信息的插入、删除、查询等功能进行实现，没有错误

**·健壮性**

程序应当对相关信息，如考生数量、考生信息等的错误输入进行处理，也要能对特殊情况（如对应的考生不存在）进行合适的处理。

**·高效性**

程序涉及到的操作有多种，并且使用都会非常频繁，这就要求要在尽可能短的时间内处理大量的数据，完成对应的操作。

**·交互性**

程序要以系统的形式出现，涉及到菜单选择、界面切换等功能，需要给用户提供足够的提示与说明，使用方式要友好。

**2.项目设计**

2.1 数据结构设计

由项目分析可以得出，本项目中需要存储大量的考生信息，同时还要频繁地对这些信息进行插入、删除、查找等操作，故需要一类数据结构，其空间占用合理，插入等操作时间复杂度不能太高。值得注意的是，考生的考号是唯一的，可以利用这个性质，使用AVL树进行存储，以此来优化存储结构，提高相关操作的性能。

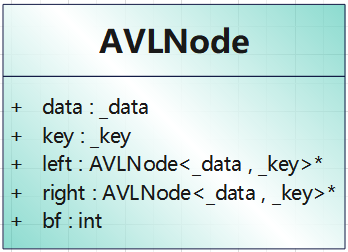
同时，统计功能对考生的报考类别进行了统计，也需要频繁地对这些信息进行插入、删除、查找等操作，但报考类别没有唯一标志，故使用链表进行存储。

考生的姓名等信息可以使用字符串来进行存储。

2.2 类设计

**2.2.1 AVL结点类（AVLNode）**

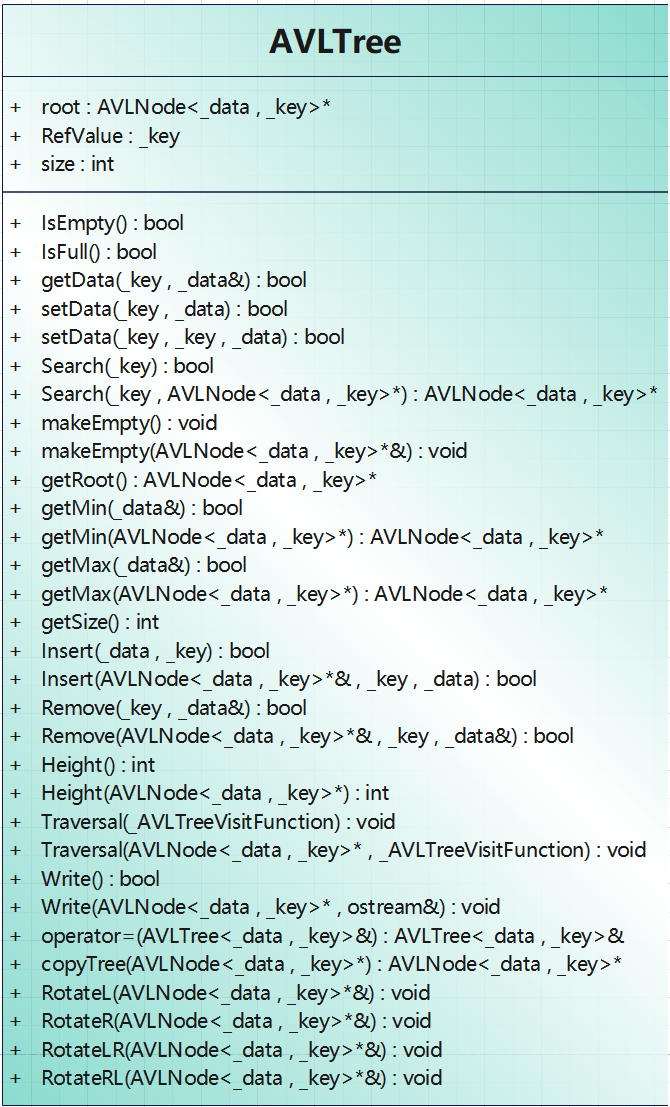
AVL结点类（AVLNode）存储了一个结点的数据域，关键码，左右子女指针以及平衡因子（即左右子树的高度差，在AVL树中用来判断是否在该结点处发生失衡），很好地描述了此节点的各种情况。其UML图如下：



**2.2.2 AVL树类（AVLTree）**

AVL树（AVLTree）是一种二叉搜索树，可以存放具有关键码的元素。与一般的搜索树区别是，其在插入和删除结点时会做平衡化旋转，改变一些结点的连接方式，最终使得每个结点的左右子树（若存在）都较为平衡，即高度相差不超过1。这样构建出来的二叉搜索树形态平衡，接近完全二叉树，使得插入、删除、搜索等操作的时间复杂度为O(log**2** n)，性能良好。同时，其采用链式存储，不需要开辟连续空间，对于空间要求也不高。

其UML图如下：



其中，主要函数如下：

//判断AVL树是否为空

bool IsEmpty()const;

//判断AVL树是否已满

bool IsFull()const;

//从AVL树中取关键码为x的元素

bool getData(const \_key x, \_data& d)const;

//设置AVL树中的元素值（不改变关键码）

bool setData(const \_key x, const \_data d);

//设置AVL树中的元素值（改变关键码）

bool setData(const \_key oldKey, const \_key newKey, const \_data d);

//判断关键码为x的元素在不在AVL树中

bool Search(const \_key x)const;

//清空AVL树

void makeEmpty();

//取AVL树的根

AVLNode<\_data, \_key>\* getRoot()const;

//取AVL树最小元素

bool getMin(\_data& d)const;

//取AVL树最大元素

bool getMax(\_data& d)const;

//取AVL树元素个数

int getSize()const;

//往AVL树中插入一个元素

bool Insert(const \_data d, const \_key x);

//从AVL树中删去一个元素

bool Remove(\_key x, \_data& d);

//求AVL树的高度

int Height()const;

//以visit的访问方式遍历整个AVL树（中序遍历）

void Traversal(\_AVLTreeVisitFunction visit)const;

//将AVL树中的元素信息写入文件中（中序顺序）

bool Write()const;

//重载函数：赋值

AVLTree<\_data, \_key>& operator=(const AVLTree<\_data, \_key>& tree);

//将AVL树进行左单旋转

void RotateL(AVLNode<\_data, \_key>\*& ptr);

//将AVL树进行右单旋转

void RotateR(AVLNode<\_data, \_key>\*& ptr);

//将AVL树进行先左后右双旋

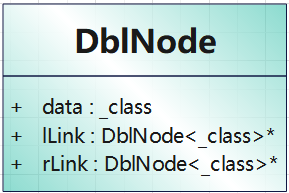
void RotateLR(AVLNode<\_data, \_key>\*& ptr);

//将AVL树进行先右后左双旋

void RotateRL(AVLNode<\_data, \_key>\*& ptr);

**2.2.3 双向链表结点类（DblNode）**

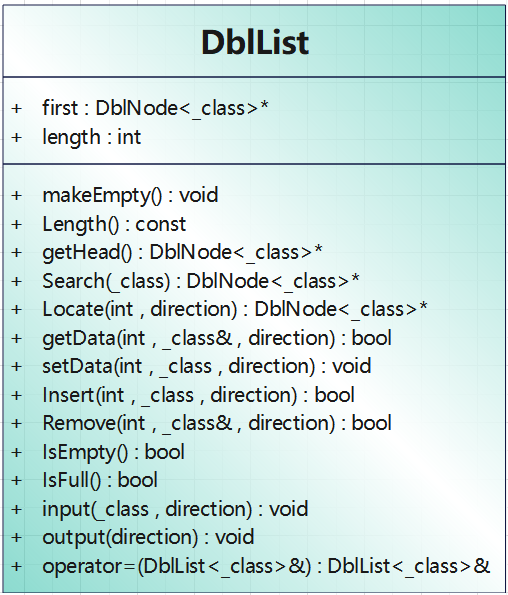
双向链表结点类（DblNode）除了存储数据，还存储了指向前驱和后继节点的指针，适用于双向链表。其UML图如下：



**2.2.4 双向链表类（DblList）**

双向链表类（DblList）是一种链表，通过指针指向前驱和后继节点。此外，为了降低插入、删除元素等操作的复杂度，附加了头结点，同时头尾指针都指向链表的另一端，形成了循环链表。

其UML图如下：



其中，主要函数如下：

//将链表置为空表

void makeEmpty();

//计算链表的长度

int Length()const;

//取附加头结点地址

DblNode<\_class>\* getHead()const;

//在链表中沿后继方向寻找等于给定值x的结点

DblNode<\_class>\* Search(const \_class x)const;

//在链表中定位到指定方向上第i (i >= 0) 个结点

DblNode<\_class>\* Locate(const int pos, const direction d)const;

//取出指定方向上第i (i > 0) 个结点的值

bool getData(const int pos, \_class& x, const direction d)const;

//用x修改指定方向上第i (i > 0) 个结点的值

void setData(const int pos, const \_class x, const direction d);

//在指定方向上第i (i >= 0) 个结点后插入一个包含有值x的新结点

bool Insert(const int pos, const \_class x, const direction d);

//删除指定方向上第i个结点，x返回其值

bool Remove(const int pos, \_class& x, const direction d);

//判断链表是否为空

bool IsEmpty()const;

//判断链表是否已满

bool IsFull()const;

//按指定方向逐个输入元素

void input(\_class endTag, const direction d);

//按指定方向逐个输出元素

void output(const direction d);

//重载函数：赋值

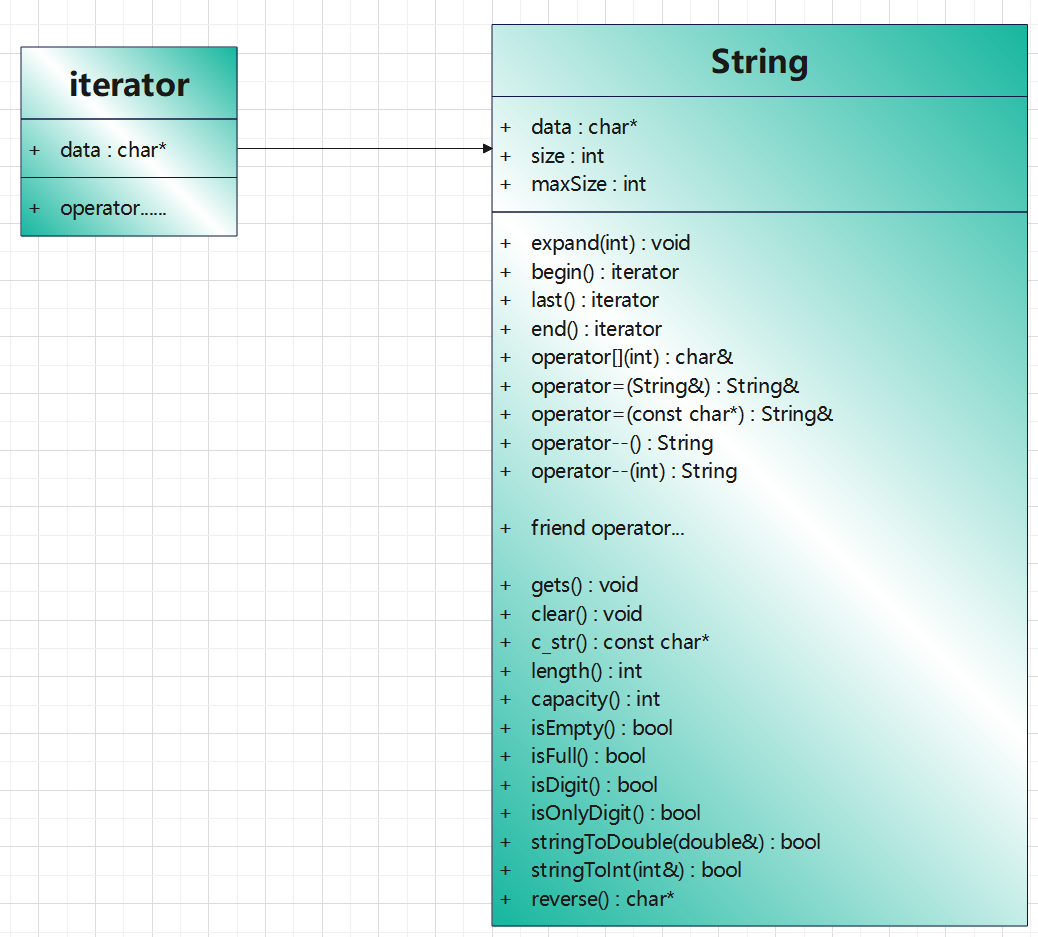
DblList<\_class>& operator=(DblList<\_class>& L);

**2.2.4 字符串类（String）**

字符串本质上是通过字符数组实现的，通过动态开辟一个数组存放各个字符。由于字符串的添加、比较、赋值等操作使用得十分频繁，故该类重载了许多运算符的操作，使得这些操作使用起来更易上手。

另一种常用的字符串操作是对其的遍历，在此添加了迭代器类（iterator）和一些运算符重载，以便对字符串的遍历等操作。由于这个类名在标准库中已有实现，且此处的迭代器类为String类特有，故使用了嵌套类的形式，使得iterator只能在String类内访问。

该类和其内部的iterator类的UML图如下所示：



其中，主要函数如下：

//增加数组空间大小

void expand(const int \_timesOfExpandingDefaultSize);

//返回字符串的起始位置

inline String::iterator begin();

inline const String::iterator begin()const;

//返回字符串末尾的后一个位置

inline String::iterator end();

inline const String::iterator end()const;

//返回字符串的末尾位置

inline String::iterator last();

inline const String::iterator last()const;

//重载函数：下标访问

char& operator[](const int pos)const;

//重载函数：复制（从String对象复制）

String& operator=(const String& str);

//重载函数：复制（从const char\*变量复制）

String& operator=(const char\* str);

//输入字符串（支持空格输入）

void gets();

//将字符串置为空

void clear();

//返回字符串首地址（以const char\*形式返回）

const char\* c\_str()const;

//返回字符串长度

int length()const;

//返回字符串最大容量

int capacity()const;

//判断字符串是否为空

bool isEmpty()const;

//判断字符串是否已满

bool isFull()const;

//判断字符串是否表示一个数值

bool isDigit()const;

//判断字符串是否仅有数字

bool isOnlyDigit()const;

//将字符串转成双精度数

bool stringToDouble(double& d)const;

//将字符串转成整型数

bool stringToInt(int& i)const;

//将字符串反转

char\* reverse();

2.3 算法设计

**2.3.1 算法思路**

由于考生的考号具有唯一性，故存储时以考号作为关键码存储。程序支持的操作较多，但大多都需要考生的考号再进行操作。故每次输入考生考号后，都执行一次对AVL树的搜索操作，此举可以应对多人同名的情况。而插入和删除等操作和AVL树的操作一样。

考生数量众多，为了输出的简洁性，当考生多于15人时，仅打印考号前15人的信息，并显示考生总数；小于15人时打印所有人信息。同时，每一次考生的添加和删除都会根据其报考类别进行统计，即在链表中寻找该类别的结点，找到则数量变更，若变更为0则删去该结点，节省空间。

**2.3.2 性能评估**

设每次操作前已有考生人数为n，报考种类为m，可分为以下三类分析：

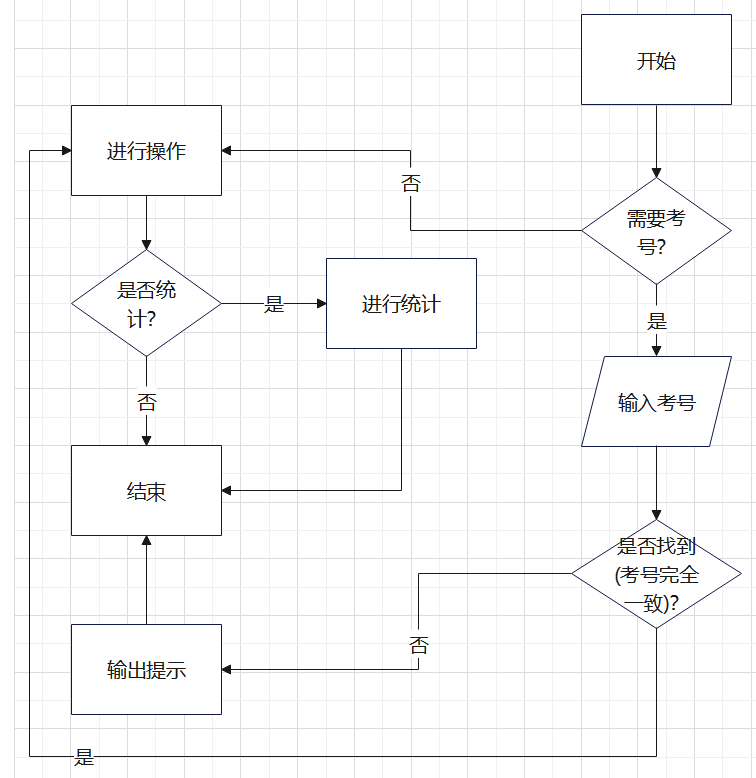
（1）仅在树中搜索：查询、不改变关键码的修改——O(log**2** n)

（2）在树中搜索+统计：插入、删除、更改关键码的修改——O(m+log**2** n)

（3）仅在链表中进行遍历：统计——O(m)

**2.3.3 一般操作过程**

·流程图表示

****

·插入考生代码实现

void System\_Insert(AVLTree<student, int>& System, DblList<typeinfo>& Info)

{

    cout << "请输入你要插入的考生的考号，姓名，性别，年龄及报考类别！" << endl;

    student stu;  /\*存放输入的考生信息的结构体\*/

    cin >> stu;   /\*输入信息\*/

    int key = 0;

    stu.num.stringToInt(key);  /\*取关键码\*/

    System.Insert(stu, key);  /\*插入AVL树中\*/

    Info\_InsertType(Info, stu.type);  /\*对报考类别进行统计\*/

    cls();

    System\_Statistic(System);

}

·删除考生代码实现

void System\_Remove(AVLTree<student, int>& System, DblList<typeinfo>& Info)

{

    if (System.IsEmpty())

        cout << "系统内无任何考生信息，请至少输入一个考生的信息！" << endl;

    else {

        String num;

        int key = 0;

        do {

            cout << "请输入要删除的考生的考号：";

            cin >> num;

        } while (!num.isOnlyDigit());

        num.stringToInt(key);

        student stu;  /\*存放删除考生信息的结构体\*/

        if (System.getData(key, stu) && stu.num == num) {  /\*若该考生存在，删除并进行统计\*/

            cout << "你删除的考生信息是：\n" << stu;

            System.Remove(key, stu);

            Info\_RemoveType(Info, stu.type);

        }

        else  /\*若该考生不存在，输出提示\*/

            cout << "该考生不存在！" << endl;

    }

    pause();

    cls();

    System\_Statistic(System);

}

·搜索考生代码实现

void System\_Search(AVLTree<student, int>& System)

{

    if (System.IsEmpty())

        cout << "系统内无任何考生信息，无法查询！" << endl;

    else {

        String num;

        int key = 0;

        do {

            cout << "请输入要搜索的考生的考号：";

            cin >> num;

        } while (!num.isOnlyDigit());

        num.stringToInt(key);

        student stu;  /\*存放考生信息的结构体\*/

        if (System.getData(key, stu))  /\*若该考生存在，输出信息\*/

            cout << "查询结果：\n" << stu;

        else  /\*若该考生不存在，输出提示\*/

            cout << "该考生不存在！" << endl;

    }

    pause();

    cls();

    System\_Statistic(System);

}

·修改考生信息代码实现

void System\_Revise(AVLTree<student, int>& System, DblList<typeinfo>& Info)

{

    if (System.IsEmpty())

        cout << "系统内无任何考生信息，无法修改！" << endl;

    else {

        String num;

        int oldKey = 0, newKey = 0;  /\*新旧关键码\*/

        do {

            cout << "请输入要修改的考生的考号：";

            cin >> num;

        } while (!num.isOnlyDigit());

        num.stringToInt(oldKey);

        student stu;  /\*存放考生原信息的结构体\*/

        String oldType, newType;  /\*新旧报考类别\*/

        if (System.getData(oldKey, stu) && stu.num == num) {

            cout << "考生原信息：\n" << stu;  /\*输出考生原信息\*/

            oldType = stu.type;  /\*取旧报考类别\*/

            cout << "请输入考生新信息：" << endl;

            cin >> stu;  /\*输入新信息\*/

            newType = stu.type;  /\*取新报考类别\*/

            stu.num.stringToInt(newKey);  /\*取新关键码\*/

            /\*根据是否改变了关键码进行更改操作\*/

            newKey == oldKey ? System.setData(newKey, stu) : System.setData(oldKey, newKey, stu);

            //if (newKey == oldKey)  /\*根据是否改变了关键码进行更改操作\*/

                //System.setData(newKey, stu);

            //else

                //System.setData(oldKey, newKey, stu);

            if (newType != oldType) {  /\*根据是否改变了报考类别进行统计操作\*/

                Info\_RemoveType(Info, oldType);

                Info\_InsertType(Info, newType);

            }

            cout << "考生信息已修改！" << endl;

        }

        else

            cout << "该考生不存在！" << endl;

    }

    pause();

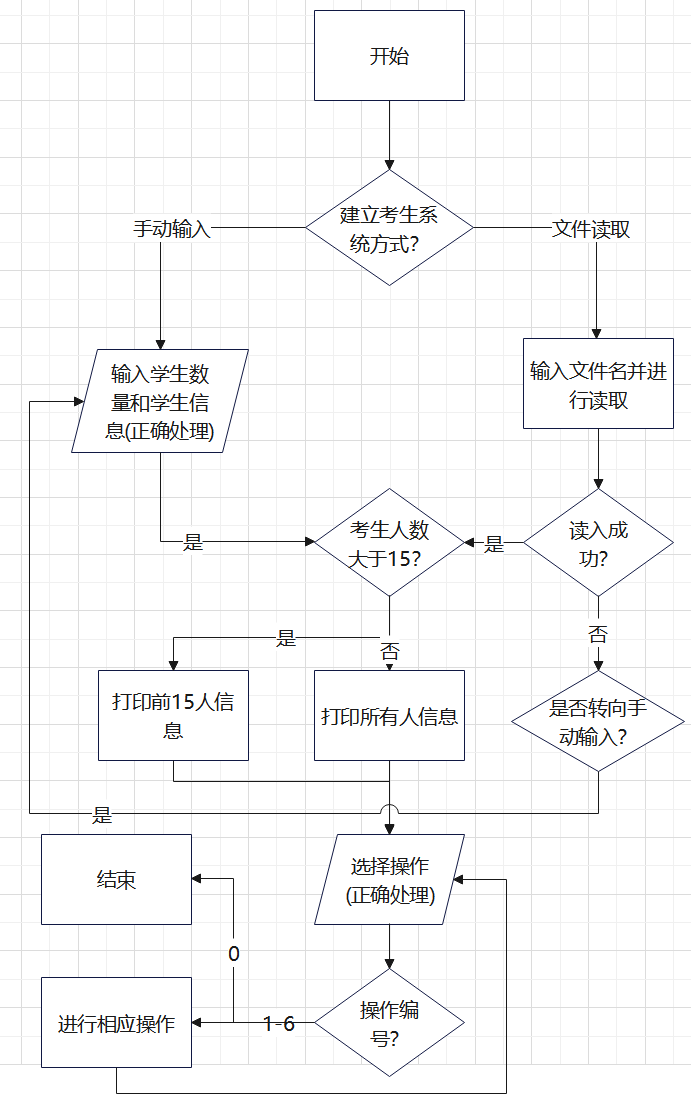
    cls();

    System\_Statistic(System);

}

**2.3.4 项目主体部分**

·流程图表示：



·代码实现：

int main()

{

    AVLTree<student, int> System;  /\*存放考生信息的AVL树\*/

    DblList<typeinfo> Info;  /\*统计链表\*/

    cout << "首先请建立考生信息系统！\n"

        << "请选择建立方式：\n"

        << "1.从文件读取\n"

        << "2.手动输入\n"

        << "您的选择是：";

    char way;

    /\*选择建立方式\*/

    while (1) {

        way = getchar();

        if (way >= '1' && way <= '2')

            break;

    }

    /\*从文件读取\*/

    if (way == '1') {

        while (1) {

            cout << "\n请输入文件名：";

            String filename;

            cin >> filename;

            cin.ignore(INT\_MAX, '\n');

            ifstream reader;

            reader.open(filename.c\_str(), ios::in);

            if (!reader) {

                cerr << "文件" << filename << "打开失败！是否重新输入(y/n)";

                char choice;

                while (1) {

                    choice = getchar();

                    if (choice == 'y' || choice == 'Y' || choice == 'n' || choice == 'N')

                        break;

                }

                if (choice == 'n' || choice == 'N') {

                    cout << "停止文件读取，已自动转向手动输入" << endl;

                    way = '2';

                    break;

                }

            }

            else {

                System\_Read(System, Info, reader);

                cout << "读取完成！" << endl;

                reader.close();

                pause();

                cls();

                System\_Statistic(System);

                break;

            }

        }

    }

    /\*手动输入\*/

    if (way == '2') {

        int num = num\_input("\n请输入考生人数：", 1, maxSize);

        cout << "请依次输入考生的考号，姓名，性别，年龄及报考类别！" << endl;

        for (int i = 0; i < num; i++) {

            student stu;

            cin >> stu;

            int key = 0;

            stu.num.stringToInt(key);

            System.Insert(stu, key);

            Info\_InsertType(Info, stu.type);

        }

        cls();

        System\_Statistic(System);

    }

    bool if\_end = false;

    while (!if\_end) {

        cout << "\n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，6为保存，0为取消操作）：";

        char selection;

        /\*选择操作\*/

        while (1) {

            selection = getchar();

            if (selection >= '0' && selection <= '6')

                break;

        }

        switch (selection) {

        case '0':

            if\_end = true;

            break;

        case '1':

            System\_Insert(System, Info);

            break;

        case '2':

            System\_Remove(System, Info);

            break;

        case '3':

            System\_Search(System);

            break;

        case '4':

            System\_Revise(System, Info);

            break;

        case '5':

            System\_Statistic(System, Info);

            break;

        case '6':

            if (System.Write())

                cout << "保存成功！" << endl;

            else

                cout << "保存未成功" << endl;

            pause();

            cls();

            System\_Statistic(System);

            break;

        }

    }

    cls();

    cout << "考生系统已关闭！" << endl;

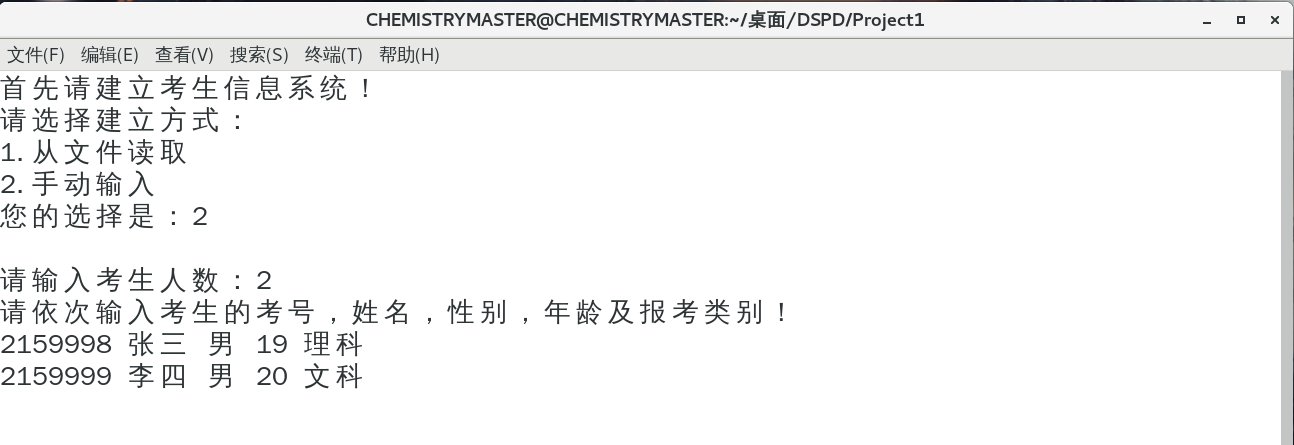
    return 0;

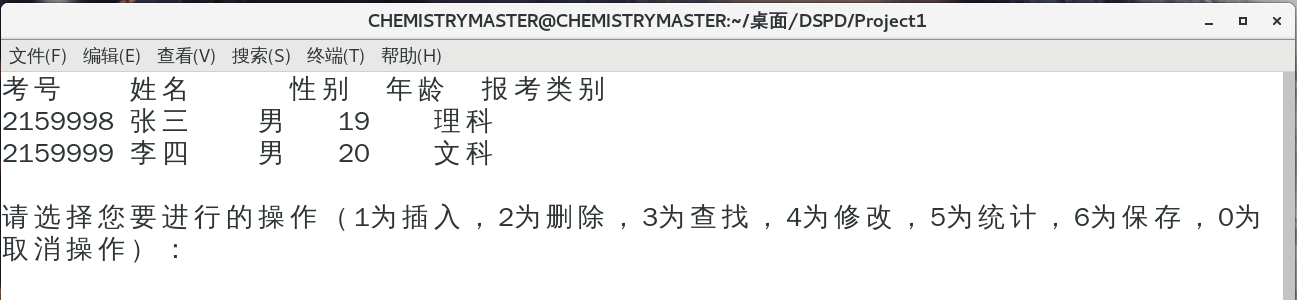
}

**3.项目测试**

3.1 建立考生系统（手动输入）

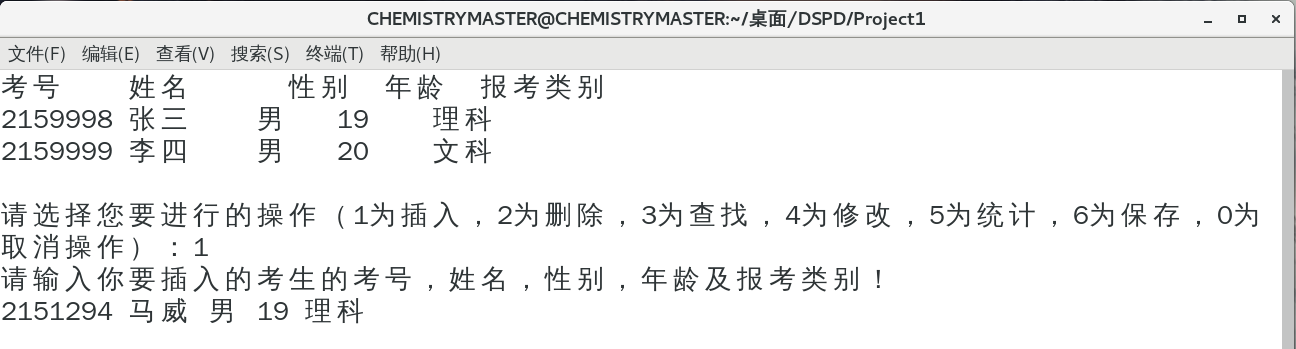
测试结果：

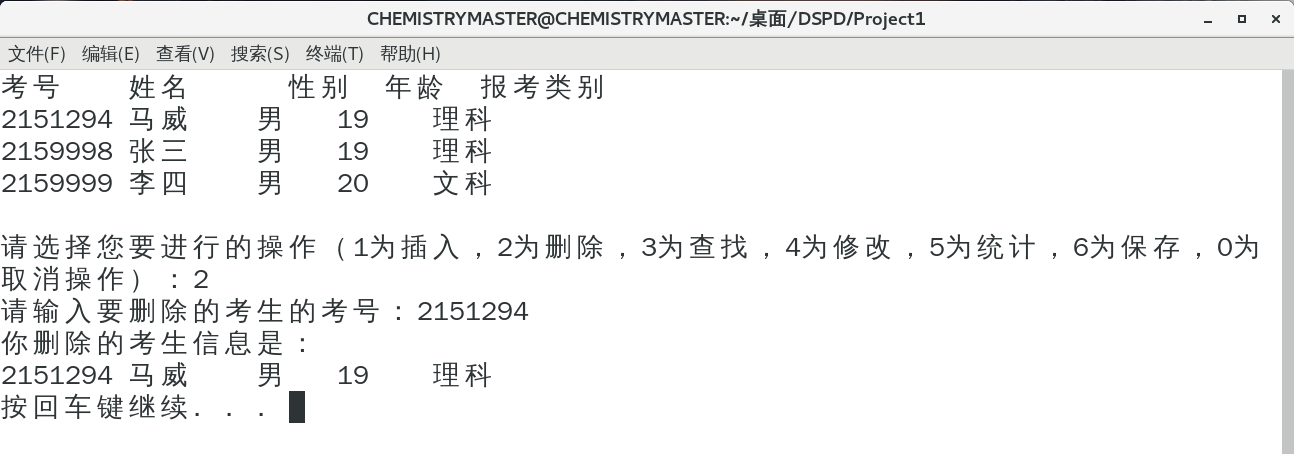




3.2 插入考生信息

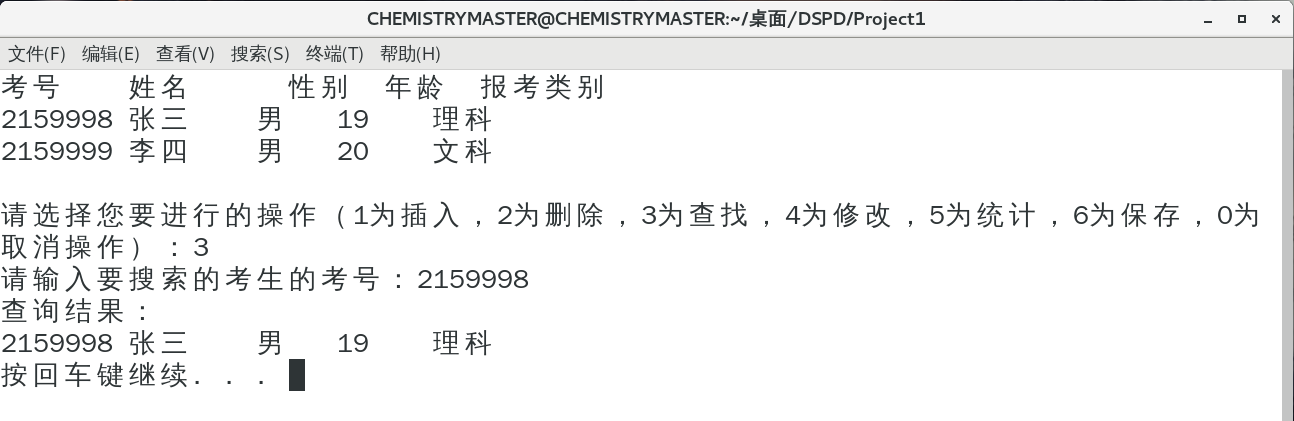
测试结果：





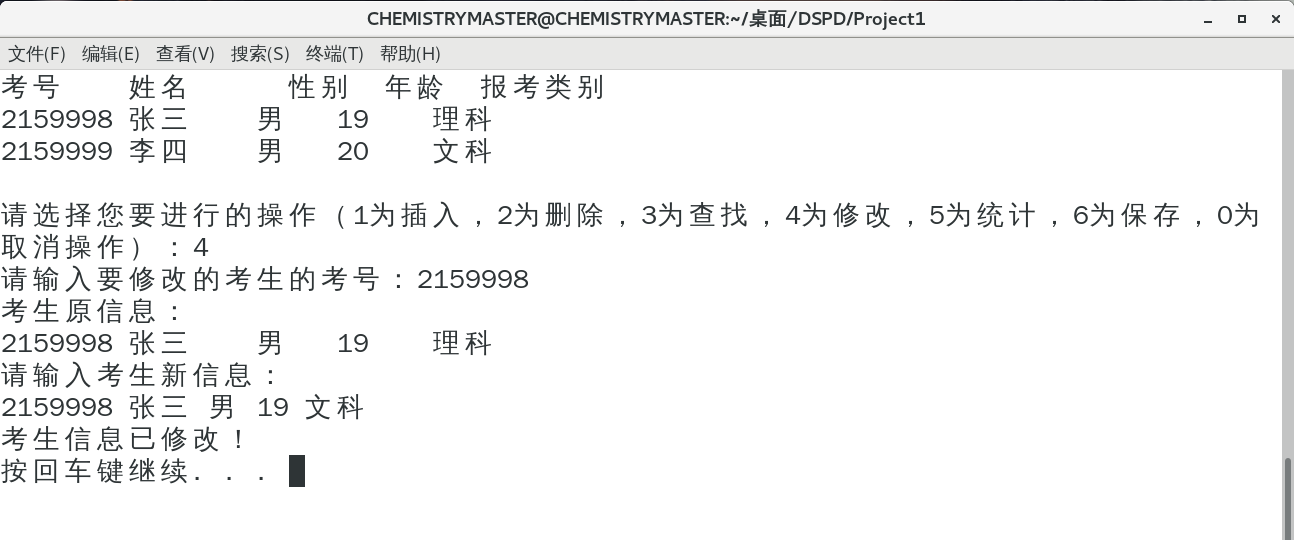
3.3 查找考生信息

测试结果：



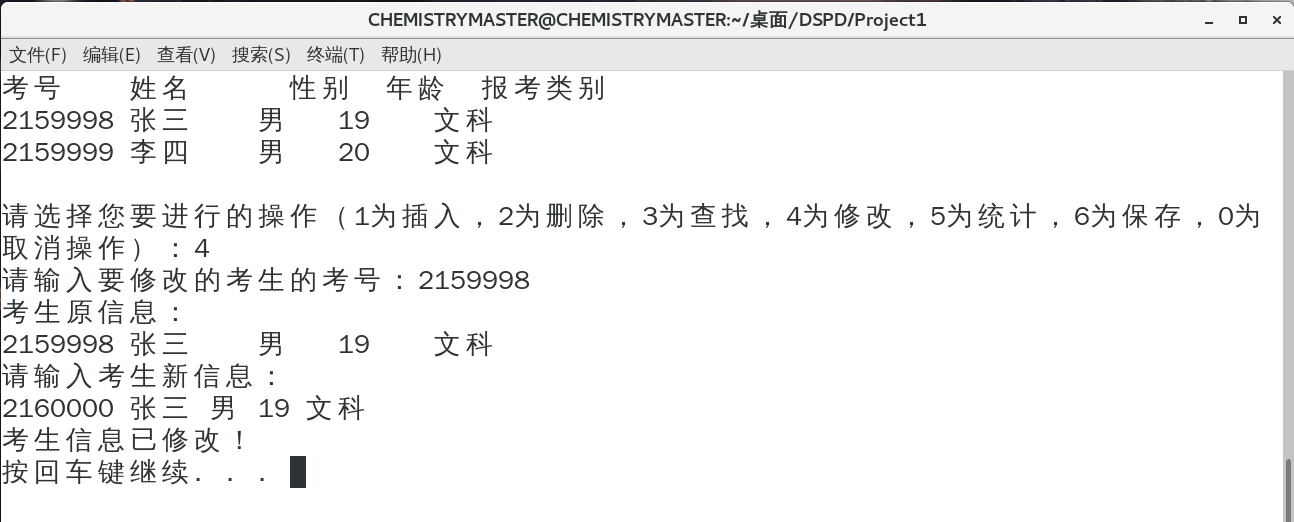
3.4 修改考生信息（不改变考号）

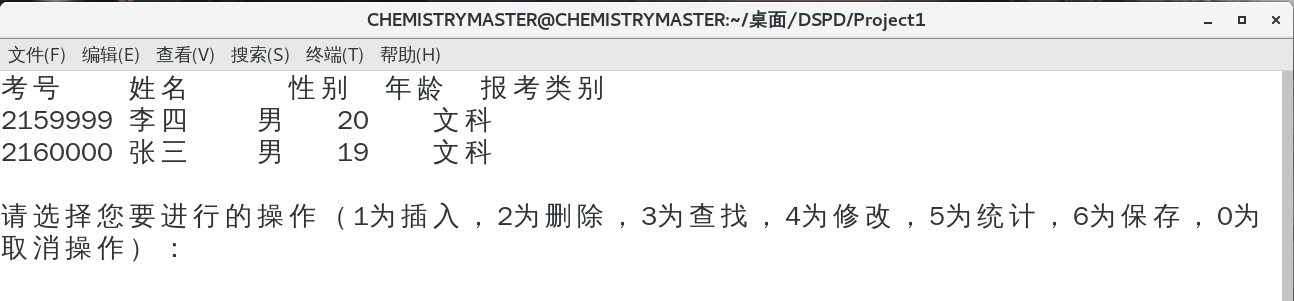
测试结果：



3.5 修改考生信息（改变考号）

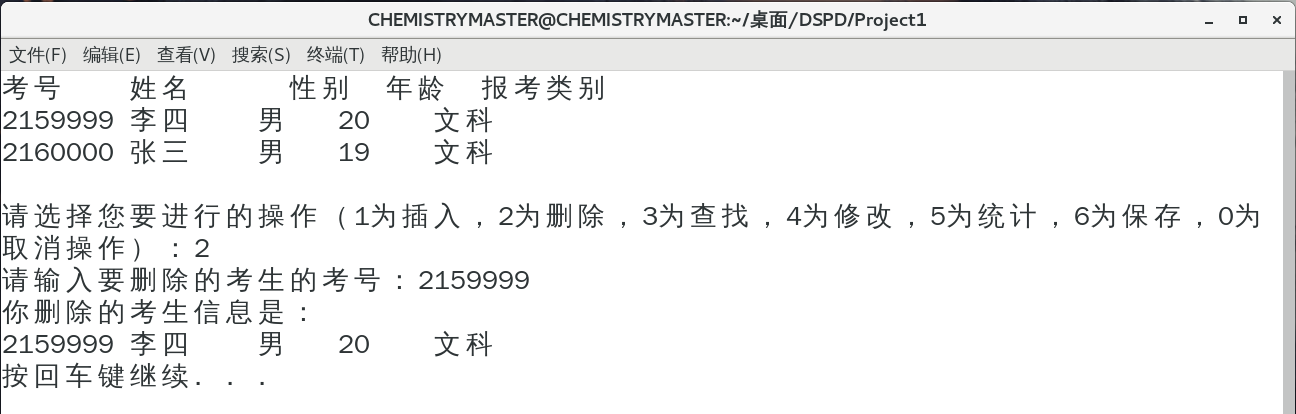
测试结果：

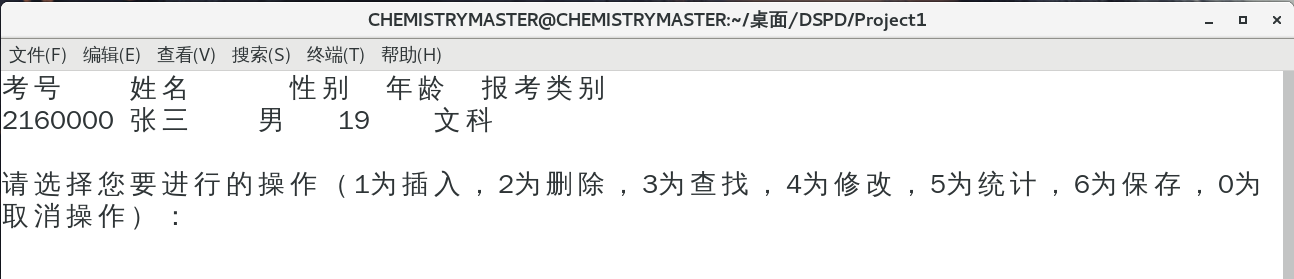




3.6 删除考生信息

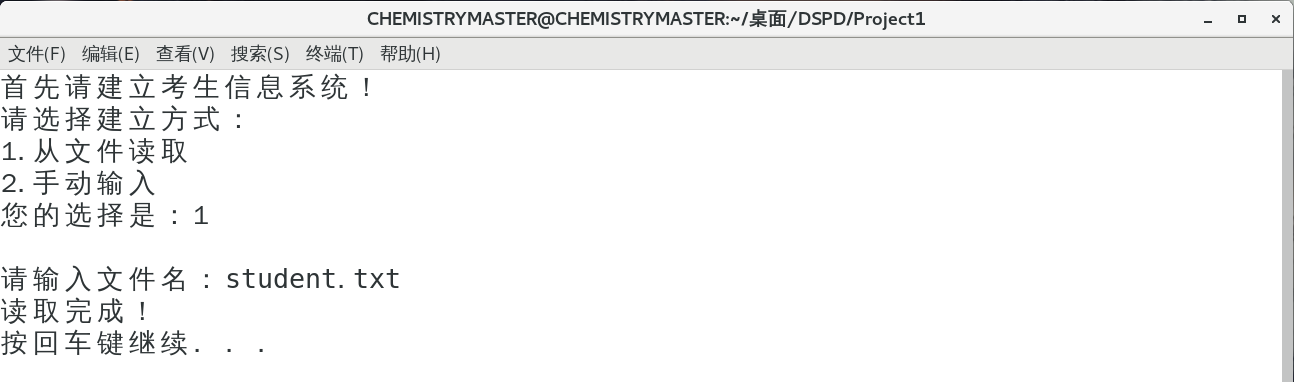
测试结果：

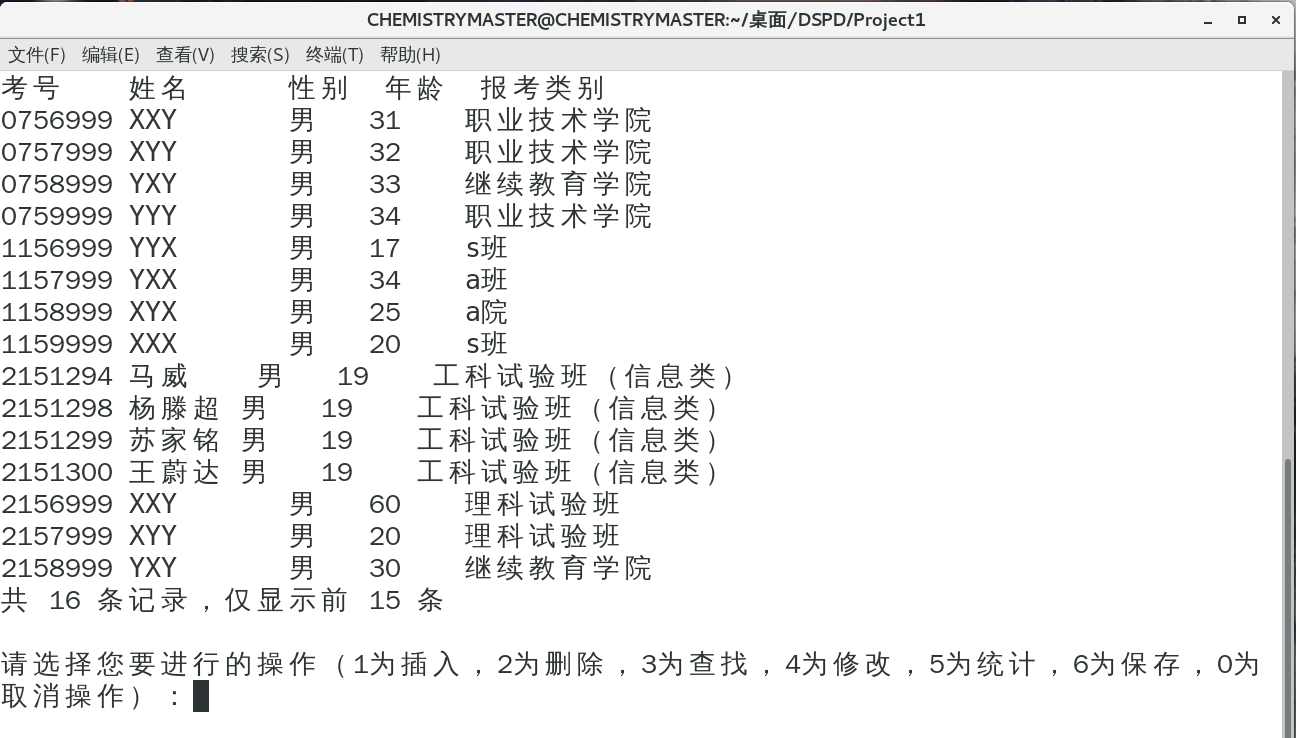




3.7 建立考生系统（从文件读取）& 考生总数大于15

测试结果：





3.8 统计考生信息

测试结果：

