项目说明文档

数据结构课程设计

——考试报名系统

作 者 姓 名： 刘畅

学 号： 2054164

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 1](#_Toc91009605)

[1.1 背景分析 1](#_Toc91009606)

[1.2 功能分析 1](#_Toc91009607)

[2 设计 1](#_Toc91009608)

[2.1 数据结构设计 1](#_Toc91009609)

[2.2 类结构设计 1](#_Toc91009610)

[2.3 成员与操作设计 2](#_Toc91009611)

[2.4 系统逻辑设计 3](#_Toc91009612)

[3 实现 4](#_Toc91009613)

[3.1 插入功能的实现 4](#_Toc91009614)

[3.1.1 实现方法 4](#_Toc91009615)

[3.1.2 插入功能核心代码（错误处理不进行展示） 4](#_Toc91009616)

[3.2 删除功能的实现 4](#_Toc91009617)

[3.2.1 实现方法 4](#_Toc91009618)

[3.2.2 删除功能核心代码 4](#_Toc91009619)

[3.3 查找功能的实现 5](#_Toc91009620)

[3.3.1 实现方法 5](#_Toc91009621)

[3.3.2 查找功能核心代码 5](#_Toc91009622)

[3.4 修改功能的实现 5](#_Toc91009623)

[3.4.1 实现方法 5](#_Toc91009624)

[3.4.2 修改功能核心代码 5](#_Toc91009625)

[3.5 打印功能的实现 6](#_Toc91009626)

[3.5.1 实现方法 6](#_Toc91009627)

[3.5.2 打印功能核心代码 6](#_Toc91009628)

[3.6 总体系统的实现 6](#_Toc91009629)

[3.6.1 实现方法 6](#_Toc91009630)

[3.6.2 总体系统核心代码 6](#_Toc91009631)

[4 测试 7](#_Toc91009632)

[4.1 功能测试 7](#_Toc91009633)

[4.1.1 建立系统功能测试 7](#_Toc91009634)

[4.1.2 添加学生功能测试 7](#_Toc91009635)

[4.1.3 打印功能测试 7](#_Toc91009636)

[4.1.4 查询功能测试 8](#_Toc91009637)

[4.1.5 修改功能测试 8](#_Toc91009638)

[4.1.6 删除功能测试 8](#_Toc91009639)

[4.2 边界错误测试 9](#_Toc91009640)

[4.2.1 输入人数错误 9](#_Toc91009641)

[4.2.2 删除后链表为空 9](#_Toc91009642)

[4.2.3 输入学生编号不存在 9](#_Toc91009643)

[4.2.4 操作码不存在 10](#_Toc91009644)

# 1 分析

## 背景分析

考试报名系统是每场考试都不可或缺的工具，在计算机快速发展的时代，我们不再需要进行纸质记录，而是转为利用计算机的某种数据结构存储各个学生的信息，做到快速添加、查找、删除等基本功能，以此处理各个学生的信息。

不仅是考试报名系统，该框架也可以使用在其他很多情况，只要是类似二维表格的数据就都能使用该数据结构存储，换句话说，“考试报名系统”背后隐藏的数据结构在今后的各种开发中会有极大的用途，因此熟练掌握这样一套结构具有十分重要的意义。

## 1.2 功能分析

作为一个最简易的考试报名系统，首先应该有的功能就是输入同学们的考试报名情况并且可以予以显示。其次，考试报名系统还应该具有插入、删除、修改功能，以保证同学可以随时更改自己的考试报名情况。最后，考试报名系统软件还应该确保软件可以正常关闭。

综上所述，一个考试报名系统至少应该具有输入、输出、插入、删除、修改、退出的功能。

# 2 设计

## 2.1 数据结构设计

考试报名系统要求大量的添加、删除、修改操作，而链表进行增加、删除等操作十分简便，因此考虑使用链表数据结构。同时，为了实现简易，在第一个结点之前附加头结点，这样就使得增加或删除头结点与处理其他结点方法相同，使得程序简洁。

## 2.2 类结构设计

链表结构包括两个类结构：链表结点类（LinkNode）和链表类（LinkedList），为了能使LinkedList类更为方便地访问LinkNode类的数据和方法，将LinkNode类都设为public类型。

将学生设计为学生模板类（Student），作为一个新的数据类型进行操作，增添其私有属性，并重载其输入输出。

为了使链表具体实现和学生模板分离，采用template模板进行链表的编写（由于以template编写的类无法在不同文件声明和实现，故将LinkNode和LinkedList声明和实现全部写在LinkedList.hpp中），再用Student作为LinkedList类型参数实例化一个学生链表对象。

最后还需要一个逻辑控制类（StudentManager）用来进行与用户的交互，即输入提示词、接受用户的操作码并调用学生链表对象的特定功能等等。

## 2.3 成员与操作设计

**链表结点类（LinkNode）**

**public：**

LinkNode() :\_data(T()), \_next(NULL);//初始化为空

T \_data;//结点数据

LinkNode<T>\* \_next;//下一结点

**链表类（LinkedList）**

**public：**

LinkedList() :\_first(new LinkNode<T>), \_length(0);//构造函数

~LinkedList();//析构函数

void Erase();//清空链表

int getLength()const;//获取链表长度

LinkNode<T>\* getHead()const;//获取附加头结点

LinkNode<T>\* Locate(int pos);//定位

bool push\_back(T& t);//在链表末尾添加结点

bool Remove(int pos);//移除结点

**private：**

LinkNode<T>\* \_first;//链表附加头结点

int \_length;//链表长度

**学生类（Student）**

/\*私有成员，包括编号、考号、姓名、年龄、报考类别，每个私有成员都设置set函数和get函数；另外将Student类的输入输出进行重载，以便更方便地对每个链表结点的数据进行操作。\*/

**public：**

friend istream& operator>>(istream& input, Student& stu);

friend ostream& operator<<(ostream& output, const Student& stu);

**界面管理类（StudentManager）**

**public：**

StudentManager();

~StudentManager();

void run();//类运行函数

**private：**

LinkedList<Student> stuList;//学生链表

void PrintInterFace();//打印提示词

void insertStudent();//添加学生

void pollStudent();//拉取学生

void modifyStudent();//修改学生

void removeStudent();//删除学生

void printStudent();//打印学生

void updateStudentId();//更新学生id

## 2.4 系统逻辑设计

系统首先调用InitScreen()函数实现对屏幕的初始化，完成对链表L的创建和输入数据工作，然后根据用户所输入的操作码（operatorCode）执行链表L对应的成员函数。

在main.cpp中实例化StudentManager对象，并调用其run()函数；在run()函数先调用insertStudent()与updateStudent()函数来建立考生信息系统，而后通过不断读取用户输入的操作码（opeNum），实现对应的功能（即建立循环，跳出条件是操作码为零）。

在输入学生信息的时候，我们需要依次输入准考证号(int)、姓名(string)、性别(char)、年龄(short int)、报考类别(string)，为了方便修改、添加、拉取（即查询），系统会将每个输入的学生依次编号，即Student类中的\_id，比如在原有3个学生的基础上再添加2个学生，编号会自动从1、2、3补足到4、5，而在修改、添加、拉取的时候，只需要直接输入学生的编号，便能轻松定位到学生，并且由于编号id是系统自行升序生成的，每个学生都有唯一的编号与之对应，不会出现重复现象。

对于每个操作（添加、拉取、修改、删除等），我们只需调用LinkedList.hpp中对应的成员函数，传入特定的参数即可，具体的实现过程在链表中体现，这样便实现了不同功能的分离，降低了耦合度。

# 3 实现

## 3.1 插入功能的实现

### 3.1.1 实现方法

在学生链表末尾添加学生：先找到链表的最后一个结点，然后将其\_next指向新结点，新结点的\_data更新为应插入的数据。

### 3.1.2 插入功能核心代码（错误处理不进行展示）

bool push\_back(T& t) {

LinkNode<T>\* cur = Locate(\_length);

//\_length是链表长度，会不断更新；Locate(int pos)是成员函数，返回链表第pos个结点

if (!cur) {return false;}

LinkNode<T>\* newNode = new LinkNode<T>;

if (!newNode) {return false;}

newNode->\_data = t;

newNode->\_next = cur->\_next;

cur->\_next = newNode;

\_length++;

return true;

}

## 3.2 删除功能的实现

### 3.2.1 实现方法

已知学生id，删除学生信息：先找到这个结点的前一个结点，将前一个结点指向的结点更改为要删除结点所指向的结点，然后delete要删除的结点。

### 3.2.2 删除功能核心代码

bool Remove(int pos) {

LinkNode<T>\* cur = Locate(pos - 1);

if (!cur || !(cur->\_next)) {

return false;

}

LinkNode<T>\* tar = cur->\_next;

cur->\_next = tar->\_next;

delete tar;

\_length--;

return true;

}

## 3.3 查找功能的实现

### 3.3.1 实现方法

已知学生id，查找该学生信息：使用链表的Locate(int pos)函数，直接定位该学生并返回对应的链表结点，提取其\_data即可。

Locate(int pos)函数的实现：以链表附加头结点作为current初始值，判断--pos是否为负，为负时跳出循环，不然则current指向下一个结点。最后返回current即可。

### 3.3.2 查找功能核心代码

LinkNode<T>\* Locate(int pos) {

if (pos < 0) {return NULL; }

LinkNode<T>\* cur = \_first;

while (cur && --pos >= 0) {

cur = cur->\_next;

}

return cur;

}

## 3.4 修改功能的实现

### 3.4.1 实现方法

已知学生id，修改该学生的信息：先定位该学生，即Locate(id)返回包含该学生的链表结点，再用新输入的学生信息，创建新的学生对象，来覆盖原来的学生对象。

由于链表本身不应该有修改链表中某个数据的功能，因此将其核心放在StudentManager类中的modifyStudent()中。

### 3.4.2 修改功能核心代码

void StudentManager::modifyStudent()

{

cout << "请输入要修改的考生编号：" << endl;

int id;

cin >> id;

LinkNode<Student>\* tar = stuList.Locate(id);

if (!id || !tar) {

cout << "不存在这名考生！" << endl;

}

else {

cout << "该考生目前信息为：" << tar->\_data << endl

<< "请输入修改后的考号、姓名、性别、年龄：" << endl;

cin >> tar->\_data;

cout << "修改成功！" << endl;

}

}

## 3.5 打印功能的实现

### 3.5.1 实现方法

先以链表附加头结点作为current初始值，不断将current指向下一届结点，并输出每个结点对应的数据，直到current指向的是NULL；由于将Student类的输出进行了重载，因此只需直接输出Student对象。

由于链表本身不应该包含打印功能（因为链表本身不知道如何输出其存放的元素），所以将核心代码放在StudentManager类中的printStudent()中。

### 3.5.2 打印功能核心代码

void StudentManager::printStudent()

{

LinkNode<Student>\* cur = stuList.getHead()->\_next;

while (cur) {

cout << cur->\_data << endl;

cur = cur->\_next;

}

}

## 3.6 总体系统的实现

### 3.6.1 实现方法

在StudentManager类的run()方法中，不断接受用户输入的操作码opeNum，当其为零时退出，不然则判断是否为1-5中的某一个，并执行对应的操作。

### 3.6.2 总体系统核心代码

void StudentManager::run()

{

cout << "首先请建立考生信息系统！" << endl;

insertStudent();

updateStudentId();

int opeNum;

do {

PrintInterFace();

cin >> opeNum;

if (opeNum == 1) {insertStudent();}

else if (opeNum == 2) {pollStudent();}

else if (opeNum == 3) {modifyStudent();}

else if (opeNum == 4) {removeStudent();}

else if (opeNum == 5) {printStudent();}

updateStudentId();

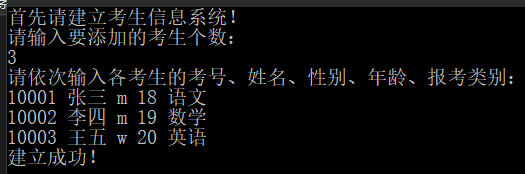
} while (opeNum != 0);

}

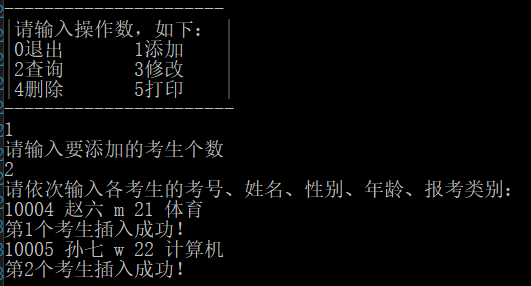
# 4 测试

## 4.1 功能测试

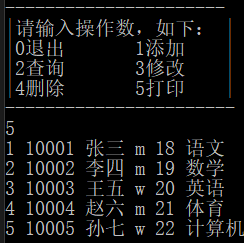
### 4.1.1 建立系统功能测试



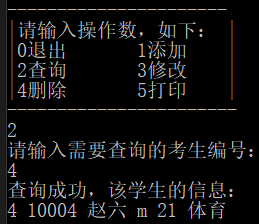
### 4.1.2 添加学生功能测试



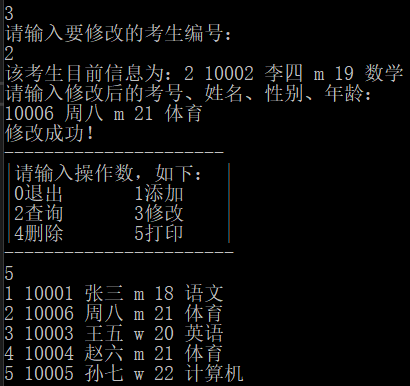
### 4.1.3 打印功能测试



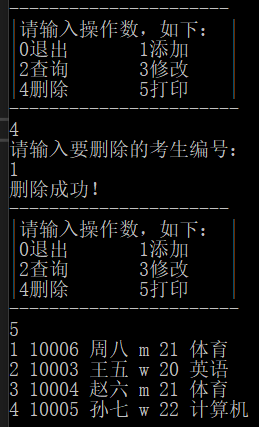
### 4.1.4 查询功能测试



### 4.1.5 修改功能测试



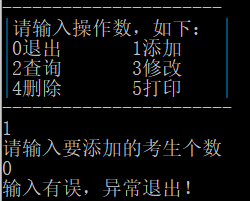
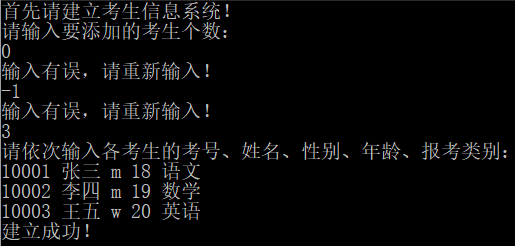
### 4.1.6 删除功能测试



## 4.2 边界错误测试

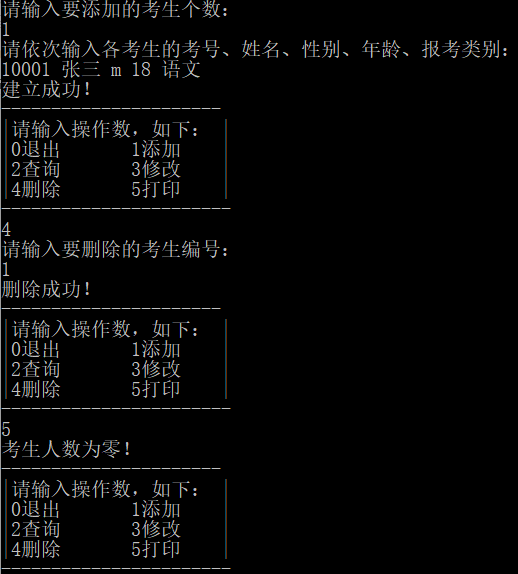
### 4.2.1 输入人数错误

预期结果：程序不奔溃，控制台出现错误提示词。



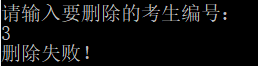
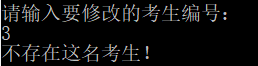
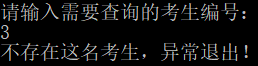
### 4.2.2 删除后链表为空

预期结果：程序正常运行，能正常进行各种操作，打印信息时能出现提示词。



### 4.2.3 输入学生编号不存在

预期结果：能提示学生编号不存在并重新输入或异常退出。



### 4.2.4 操作码不存在

