项目说明文档

数据结构课程设计

——考试报名系统

作 者 姓 名：陆诚彬

学　　号：2254321

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 项目背景

在当前的教育环境中，考试报名是一个重要且复杂的任务，特别是在高等教育机构中。随着学生数量的增加和教育体系的不断发展，传统的考试报名方法已经不能满足现代教育的需要。传统方法通常包括手工处理报名表，这不仅耗时而且容易出错。这种方式对教务管理部门造成了巨大压力，增加了工作量，同时也影响了工作效率。

因此，开发一个自动化的考试报名系统变得非常必要。这样的系统可以简化报名流程，提高数据处理的准确性和效率。本项目旨在设计并实现一个考试报名管理系统，通过电子化和自动化的方式，优化报名流程，减轻教务管理部门的工作负担，同时为学生提供更加便捷、快速的报名体验。

2 项目需求分析

2.1 功能需求

2.1.1 考生信息管理

系统应能够输入、输出、查询、添加、修改和删除考生的信息。考生信息包括但不限于准考证号、姓名、性别、年龄和报考类别。

2.1.2 数据结构设计

在设计系统时，首先要确定合适的数据结构来存储和管理考生信息。考虑到数据操作的灵活性和效率，建议使用链表作为主要的数据结构。

2.1.3 类的定义与实现

系统需要定义一个或多个类，包含成员变量和成员函数，以支持上述的信息管理功能。这包括但不限于信息录入、查询、修改、删除等相关操作的函数。

2.1.4 用户界面

虽然本项目以控制台应用程序为主，但应提供直观易用的用户界面，通过选项和命令的方式与用户互动，以便于用户完成各类操作。

2.1.5 稳定性与可靠性

系统应具备一定的容错能力，确保在错误输入或异常情况下不会崩溃，并给出适当的错误提示。

2.2 非功能需求

2.2.1 性能要求

系统应能快速响应用户的请求，处理大量数据时不会显著降低性能。

2.2.2 可维护性

代码应具有良好的注释和文档，确保今后的维护和升级方便。

2.2.3 扩展性

设计时考虑到未来可能的功能扩展和改动，使系统具备一定的灵活性。

2.2.4 用户体验

尽管是控制台应用程序，也要考虑用户体验，确保操作流程合理且易于理解。

2.3 项目输入输出需求

2.3.1 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）。

2.3.2 输出格式

输出查找学生的信息。

2.3.3 项目示例

3 项目设计

3.1 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。

3.2 类设计

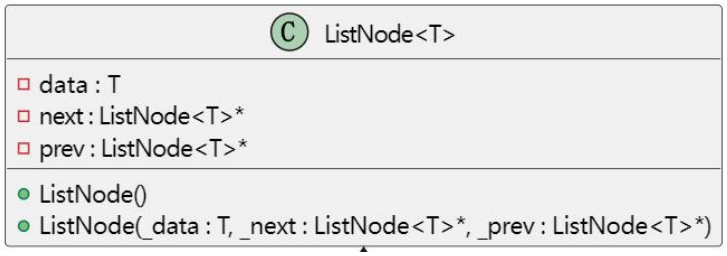
经典的链表一般包括两个抽象数据类型（ADT）——链表结点类（ListNode）与链表类（List），而两个类之间的耦合关系可以采用嵌套、继承等多种关系。

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)

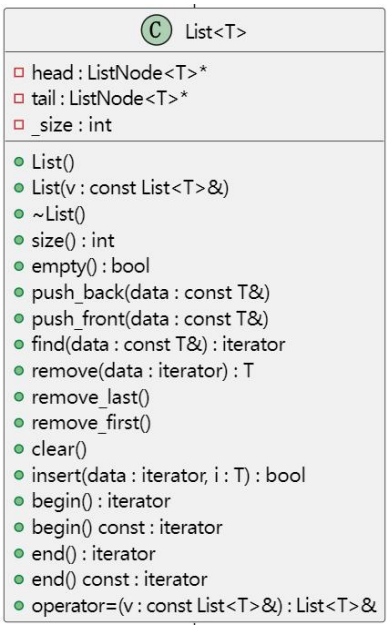
3.2.1 链表节点类（ListNode）

结点类 ListNode 用于表示双向链表中的一个结点。每个结点包含了存储的数据、指向后继结点的指针和指向前驱结点的指针。其 UML 图如下所示：

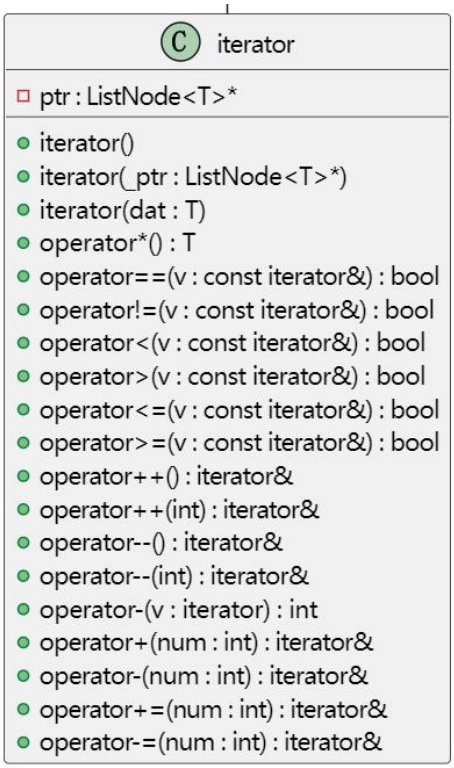


3.2.2 双向链表类（List）

双向链表类 List 实现了一系列对链表的操作。它包含头结点和尾结点，使得在链表的头部和尾部进行插入和删除操作时效率更高。其 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。



在 List 类中实现的主要功能函数如下所示：

```
List()
```
默认构造函数，用于创建一个空的链表。
```
List(const List<Type>& v)
```
拷贝构造函数，用于创建一个新的链表，其内容是另一个链表的副本。
```
~List()
```

析构函数，用于释放链表占用的所有资源。

`int size() const`

返回链表中结点的个数。

`bool empty() const`

判断链表是否为空（即链表中结点的个数是否为 0）。

`void push_back(const Type& data)`

在链表尾部插入一个新的元素。

`void push_front(const Type& data)`

在链表头部插入一个新的元素。

`iterator find(const Type& data) const`

在链表中查找值为 data 的元素，返回一个指向该元素的迭代器。

`Type remove(iterator index)`

移除迭代器所指位置的元素，并返回被移除的元素的值。

`void remove_last()`

移除链表末端的元素。

`void remove_first()`

移除链表首端的元素。

`void clear()`

清空链表，移除所有元素。

`iterator begin()`

返回一个迭代器，指向链表的第一个元素。

`iterator end()`

返回一个迭代器，指向链表的尾部（尾结点后的空结点）。

`List<Type>& operator=(const List<Type>& v)`

赋值运算符，用于将一个链表的内容复制到另一个链表中。

3.2.3 单个学生类（Student）

Student 类用于表示考试报名系统中的一个考生。每个考生包含了考号、姓名、性别、年龄和报考类别等基本信息。其功能和方法如下所示：

- **属性：**

  `int id`：考生的考号。

  `string name`：考生的姓名。

  `enum { male, female } sex`：考生的性别。

  `int age`：考生的年龄。

  `string type`：考生的报考类别。

- **构造函数：**

  `Student()`：默认构造函数，初始化考生信息。

  `Student(int ids)`：参数化构造函数，初始化考号。

- **输入输出重载：**

  `friend istream& operator>>(istream& is, Student& stu)`：重载输入流操作符，用于从流中读取考生信息。

  `friend ostream& operator<<(ostream& os, const Student& stu)`：重载输出流操作符，用于向流中写入考生信息。

◆ 方法：

`void inputData_noId(istream& is, Student& stu)`：输入除考号外的其他考生信息。

`bool operator==(const Student& stu)`：重载等于操作符，用于比较两个考生是否相等（主要根据考号）。

3.2.4 总学生数据库类（StudentList）

StudentList 类用于管理和操作考生信息的集合。它使用 List<Student> 类型的成员变量 stuList 来存储考生信息。其功能和方法如下所示：

◆ **构造函数和析构函数：**

`StudentList()`：默认构造函数。

`~StudentList()`：析构函数。

◆ **基本操作方法：**

`void uniInput(int& id)`：通用输入函数，用于输入有效的考号。

`void init()`：初始化考生信息系统，录入考生信息。

`void print()`：打印所有考生的信息。

`void find()`：查找特定考号的考生信息。

`void insert()`：插入新的考生信息。

`void erase()`：删除特定考号的考生信息。

`void update()`：更新特定考号的考生信息。

`void stat()`：统计考生信息（例如性别比例、总人数等）。

`int size()`：返回当前系统中考生的总数。

`void wait()`：暂停操作，等待用户输入。

◆ **私有方法：**

`void insertProperPlace(Student& stu)`：按照考号顺序插入考生信息。

通过这两个类的设计，考试报名系统能够有效地管理考生的信息，提供添加、删除、查找、更新和统计等功能，使得考试报名的管理工作更加高效和便捷。

4 项目实现

4.1 考生数据库初始化建立实现

4.1.1 考生数据库初始化建立功能简介

首先提示用户输入考生的总数。用户输入有效的考生数后，程序会要求用户依次输入每位考生的考号、姓名、性别、年龄和报考类别。对于每个输入的考生，程序会检查考号是否已存在于列表中。如果考号唯一，考生信息将被加入到列表中。最后，程序打印出所有已录入的考生信息，并通知用户录入完成。

4.1.2 考生数据库初始化建立功能核心代码

```
1.    void StudentList::init() {
2.        int stuCnt = 0;
3.        cout << "首先请建立考生信息系统！" << endl;
4.        cout << "请输入考生人数：";
5.        uniInput(stuCnt);
6.        if (stuCnt > 0) {
7.            cout << "请依次输入考生的考号、姓名、性别、年龄及报考类别！
   " << endl;
8.            for (int i = 0; i < stuCnt; i++) {
9.                Student stu;
10.               cin >> stu;
11.               if (stuList.find(stu) != stuList.end()) {
12.                   cout << "该考号的考生已存在，请重新输入！" << endl;
13.               }
14.               else {
15.                   this->insertProperPlace(stu);
16.               }
17.           }
18.           this->print();
19.           cout << "考生信息录入完毕！" << endl;
20.       }
21.   }
```

4.2 考生信息插入实现

4.2.1 考生信息插入功能简介

此功能允许用户将新的考生信息添加到系统中。它首先要求用户输入新考生的考号，然后输入该考生的其他信息，包括姓名、性别、年龄和报考类别。

4.2.2 考生信息插入核心代码

```
1.    void StudentList::insert() {
2.        cout << "请输入要插入的考生的考号：";
3.        int id;
4.
5.        uniInput(id);
6.
7.        Student stu(id);
8.        if (stuList.find(stu) != stuList.end()) {
9.            cout << "该考号的考生已存在！" << endl;
10.       }
11.       else {
```

```
            cout << "请依次输入考生的姓名、性别、年龄及报考类别！" << endl;
            stu.inputData_noId(cin, stu);
            this->insertProperPlace(stu);
            this->print();
            cout << "考生信息插入完毕！" << endl;
        }
    }
```

4.3 考生信息删除实现

4.3.1 考生信息删除功能简介

此功能使用户能够通过指定的考号来删除相应的考生信息。删除操作之前，系统会先确认该考号对应的考生信息确实存在于系统中。

4.3.2 考生信息删除核心代码

```
    void StudentList::erase() {
        cout << "请输入要删除的考生的考号：";
        int id;
        uniInput(id);

        auto it = stuList.find(id);
        if (it == stuList.end()) {
            cout << "不存在考号为" << id << "的考生！" << endl;
        }
        else {
            cout << "您删除的考生信息是：" << (*it).id << endl;
            stuList.remove(it);
            cout << "考生信息删除成功！" << endl;
        }
    }
```

4.4 考生信息修改实现

4.4.1 考生信息修改功能简介

此功能允许用户通过考号查找特定考生，并更新其姓名、性别、年龄和报考类别等信息。修改操作之前，系统会先确认该考号对应的考生信息确实存在于系统中。

4.4.2 考生信息修改核心代码

```
1.    void StudentList::update() {
2.        cout << "请选择您要修改的考生的考号：";
3.        int id;
4.        uniInput(id);
5.        auto it = stuList.find(id);
6.        if (it == stuList.end()) {
7.            cout << "不存在考号为" << id << "的考生！" << endl;
8.            return;
9.        }
10.        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;
11.       Student stu(id);
12.       stu.inputData_noId(cin, stu);
13.       (*it) = stu;
14.   }
```

4.5 考生信息查找实现

4.5.1 考生信息查找实现功能简介

用户可以通过输入一个特定的考号来查找对应的考生信息。如果系统中存在该考号对应的考生信息，该信息将被显示给用户；如果不存在，系统会通知用户。

4.5.2 考生信息查找实现核心代码

```
1.    void StudentList::find() {
2.        cout << "请输入要查找的考生的考号：";
3.        int id;
4.        uniInput(id);
5.
6.        auto findRes = stuList.find(id);
7.        if (findRes == stuList.end()) {
8.            cout << "不存在考号为" << id << "的考生！" << endl;
9.        }
10.       else {
11.           cout << *findRes << endl;
12.       }
13.       wait();
14.   }
```

4.6 考生信息统计实现

4.6.1 考生信息统计实现功能简介

此功能允许用户获取系统中所有考生的总数以及按性别分类的考生数量。它为用户提供了考生数据库的一个快照，显示男性和女性考生的数量。

4.6.2 考生信息统计实现核心代码

```
1.    void StudentList::stat() {
2.        int sizeStu = stuList.size();
3.        int sizeMale = 0, sizeFemale = 0;
4.        if (sizeStu == 0)
5.        {
6.            cout << "当前系统内暂无考生！" << endl;
7.            return;
8.        }
9.        for (auto i = stuList.begin(); i != stuList.end(); i++) {
10.           if((*i).sex == Student::male) sizeMale++;
11.           else sizeFemale++;
12.       }
13.        cout << "当前系统内共有" << sizeStu << "名考生，其中男生" << sizeMale << "名，女生" << sizeFemale << "名。" << endl;
14.       return;
15.   }
```

4.7 考生信息输入重要辅助函数实现

4.7.1 考生输入重要辅助函数实现功能简介

通过以下两个函数，可以保证用户输入系统的考生信息唯一性。

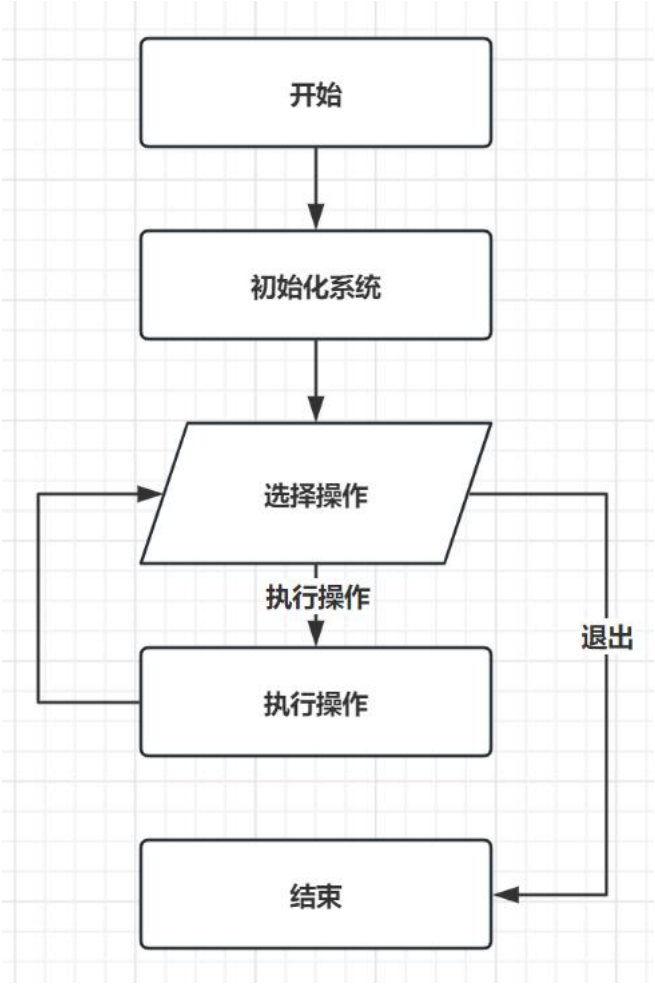
4.7.2 考生输入重要辅助函数实现核心代码

```
1.    void Student::inputData_noId(istream& is, Student& stu) {
2.        string sexStr;
3.        is >> stu.name >> sexStr >> stu.age >> stu.type;
4.        if (sexStr == "男")
5.            stu.sex = stu.male;
6.        else if (sexStr == "女")
7.            stu.sex = stu.female;
8.
9.        return;
10.   }
```

```
11.  void StudentList::insertProperPlace(Student& stu) {
12.      if (stuList.empty()) {
13.          stuList.push_back(stu);
14.      }
15.      else {
16.          for (auto i = stuList.begin(); i != stuList.end(); i++) {

17.              if ((*i).id > stu.id) {
18.                  stuList.insert(i, stu);
19.                  return;
20.              }
21.          }
22.          stuList.push_back(stu);
23.      }
24.  }
```

4.8 系统总体功能流程图



5 设计小结

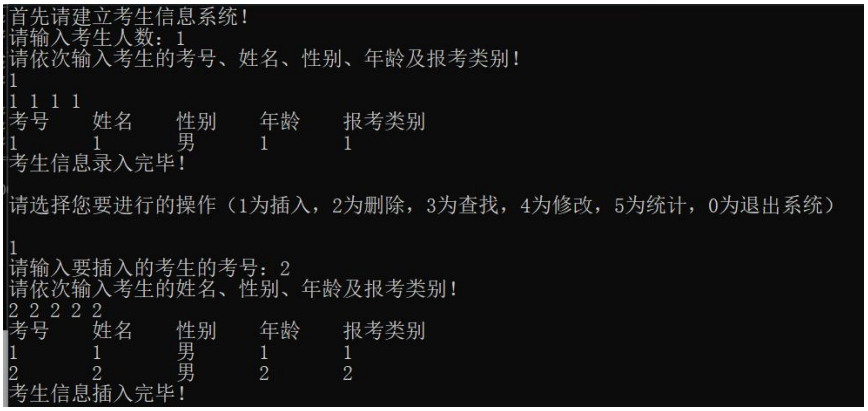
本项目成功设计并实现了一个高效的考试报名系统。通过精心的数据结构设计和类的实现，系统能够高效地处理考生信息的管理。以下是项目的主要设计亮点和实现成果：

1) **数据结构的选用：**系统核心采用双向链表数据结构，为考生信息的添加、删除、查找和修改提供了高效的支持。双向链表的选择显著提高了数据操作的灵活性和效率，特别是在大量数据的情况下。
2) **类设计与实现：**系统中实现的类（包括链表节点类、双向链表类、学生类和学生列表类）为考生信息的管理提供了强大支持。每个类都有其明确的职责，使得代码结构清晰，易于维护和扩展。
3) **用户界面友好：**尽管是基于控制台的应用程序，本系统提供了直观且易用的用户界面。用户可以轻松地执行各种操作，如输入、查询、修改、删除考生信息等，用户体验得到有效考虑。
4) **稳定性与可靠性：**系统具备良好的容错能力，能够有效地处理错误输入和异常情况，确保系统稳定运行。
5) **性能优化：**考虑到性能要求，系统对关键操作进行了优化，保证了快速响应和高效数据处理，即使是在处理大量数据时也不会出现性能下降。
6) **代码维护与扩展性：**项目代码具有良好的注释和文档，便于未来的维护和升级。在设计时已考虑到可能的功能扩展，确保了系统的灵活性和可扩展性。
7) **实用性与教育意义：**本系统不仅在实际应用中具有重要价值，同时也为数据结构课程设计提供了一个实际的应用案例，有助于加深学生对数据结构和面向对象编程的理解。

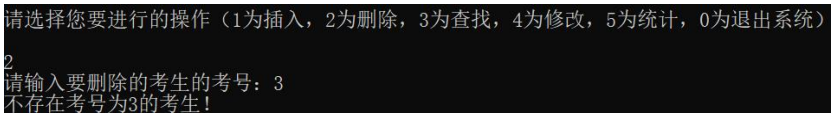
总结来说，本项目通过综合运用数据结构知识和面向对象编程技巧，成功实现了一个功能全面、性能优异且用户友好的考试报名系统，充分满足了项目的初衷和设计要求。

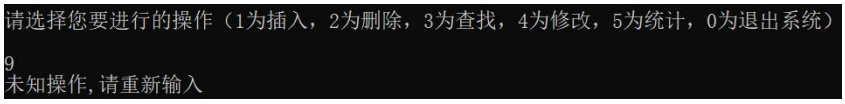
6.1 输入测试

6.1.1 正常输入



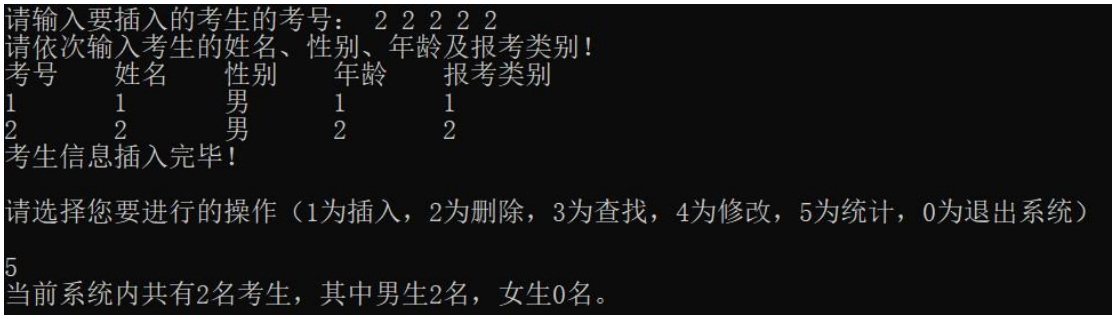
6.1.2 输入超界/非法





结论：符合输入逻辑判断

6.2 输出测试



结论：符合输出逻辑，且正确。