数据结构课程设计

项目说明文档

考试报名系统

|  |  |
| --- | --- |
| 作者姓名： | 杨烜赫 |
| 学 号： | 2252709 |
| 指导教师： | 张 颖 |
| 学院专业： | 软件学院 软件工程 |



同济大学

Tongji University

1. **题目简介**
2. **实验背景**

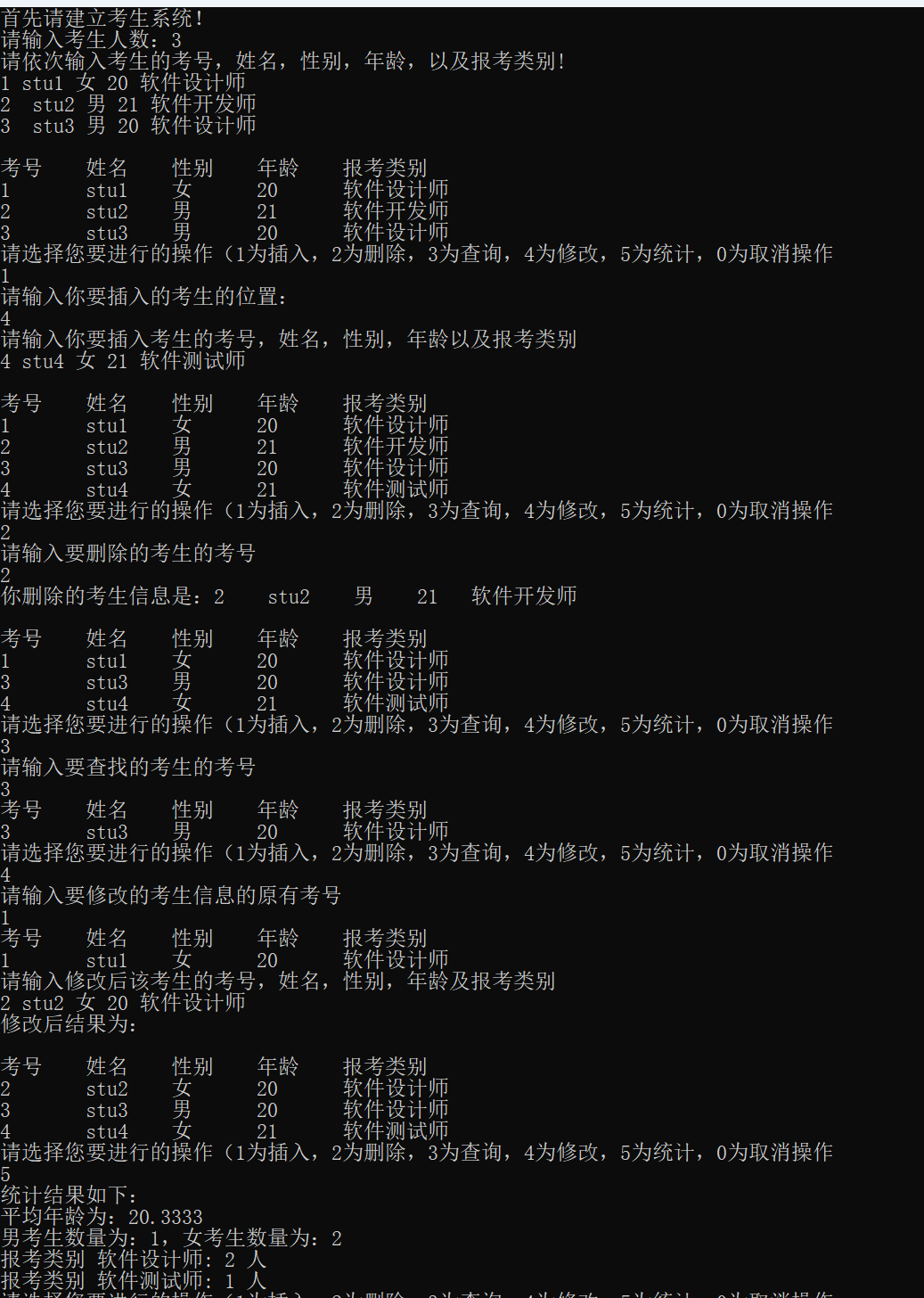
考试报名系统是一个学校不可缺少的部分，它对于学校的管理者和学生来说都至关重要，随着学生数量和考试数量的日益庞大，如何管理如此庞大的数据显得极为复杂，传统的手工管理工作量大且容易出错。

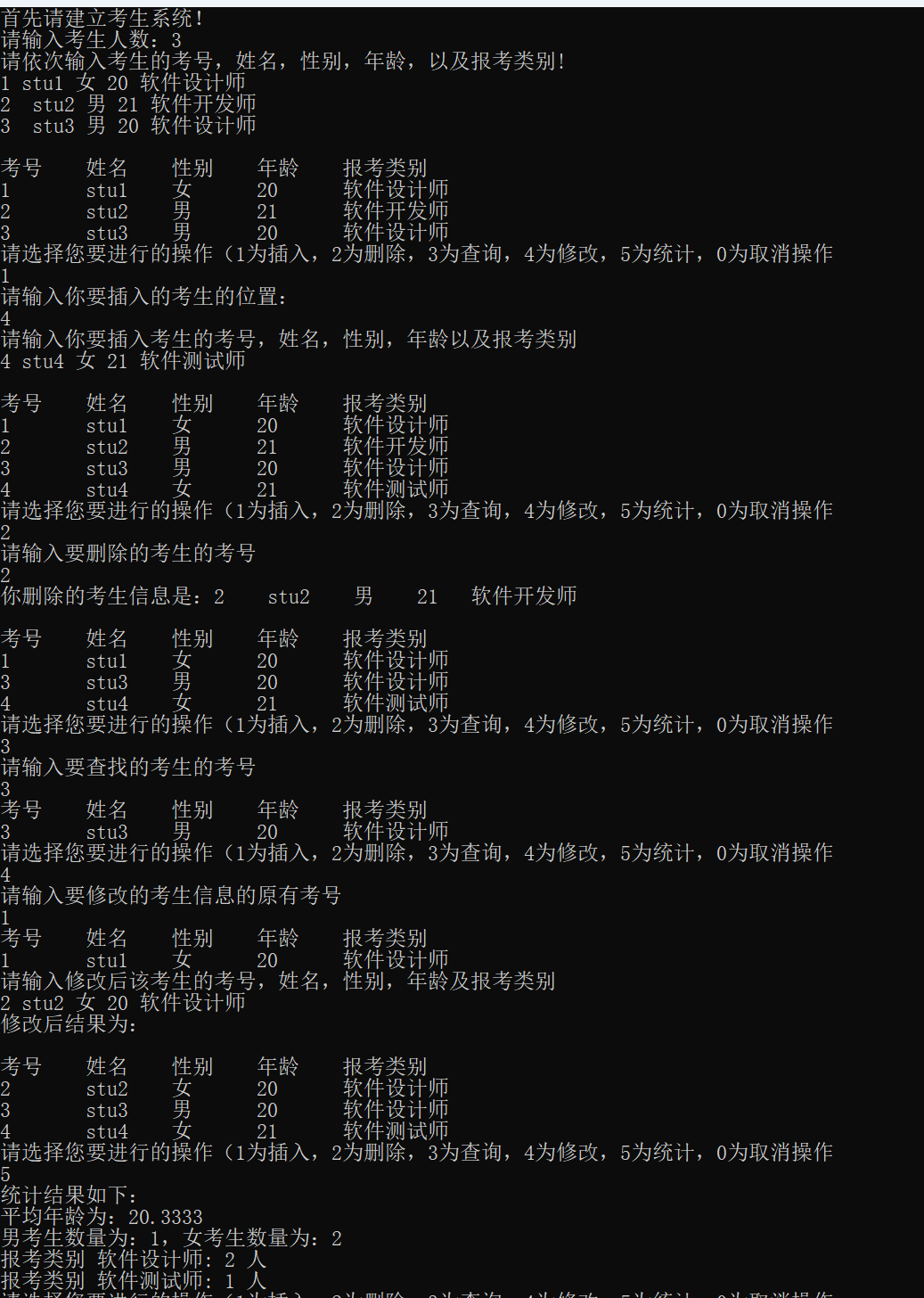
使用计算机对考试报名系统进行管理，具有手工管理所无法比拟的优势。这些优点能够极大地提高学校和学生的效率，也是学校走向信息化、科学化、国际化的重要条件。因此，开发一套考试报名系统具有十分重要的意义。

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了：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、删除考生信息等操作。

1. **实验功能**

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）

1. **项目示例**



1. **项目设计**

## 1.数据结构设计

本项目的核心目标是实现对考生信息的建立、查找、插入、修改和删除等功能。考虑到涉及大量的删除和插入操作，使用数组存储的效率并不高。为了优化这些操作，决定采用链表作为数据结构。链表具有动态存储能力，可以高效地处理插入和删除操作，特别适合于本项目的需求。

为了进一步简化操作，链表将包含一个头结点，头结点不存储实际考生信息，但作为链表的起始点，方便进行各种操作。此外，链表的尾部将设置一个空结点，这样做可以简化在链表末尾添加新节点的处理。通过这种设计，链表能够更灵活、更高效地处理考生信息。

## 2.类结构设计

在本项目中，我定义了两个类：一个节点类 (link\_node) 和一个链表类 (List)。link\_node 类用于表示链表中的每个节点，包含考生的各种信息（如考号、姓名、性别、年龄和报考类别）以及指向下一个节点的指针。这些信息和指针是链表操作的基础。

List 类是主要的功能类，用于管理整个链表，实现增加、删除、查找、修改和统计考生信息的功能。将链表类定义为节点类的友元类，使得链表类可以访问和操作节点类的私有成员，这为链表的操作提供了方便。考虑到系统的数据类型是确定的，且操作相对固定，没有采用抽象数据类型（如模板类）。直接定义了特定的数据类型，以便专注于实现项目的具体要求。

通过这种类结构设计，可以有效地建立和管理一个高效、灵活的考生信息链表，满足项目的各项功能需求。

## 3.成员与操作设计

在考生信息管理系统的设计中，类的成员和操作是核心部分。以下是具体的成员与操作设计：

**2.3.1 link\_node 类成员和操作**

成员变量：

int id：存储考生的考号。

char name[50]：存储考生的姓名。

char gender[10]：存储考生的性别。

int age：存储考生的年龄。

char category[50]：存储考生的报考类别。

link\_node\* next：指向下一个节点的指针。

构造函数：

初始化所有成员变量，尤其是指针 next 设置为 nullptr。

**2.3.2 List 类成员和操作**

成员变量：

link\_node\* first：指向链表的头结点。

int stu\_num：存储链表中的考生数量。

构造函数：

初始化链表，创建头结点，将 stu\_num 设置为0。

析构函数：

清空链表，释放所有节点占用的内存。

操作：

void make\_empty()：清空链表，删除所有节点。

bool Init\_linklist(int n)：初始化链表，创建 n 个考生的信息。

void Print(link\_node\* head)：打印链表中的所有考生信息。

bool Research()：根据考号查找考生信息。

bool Insert()：在链表中插入新的考生信息。

bool Remove()：根据考号删除考生信息。

bool Modify()：修改指定考号的考生信息。

bool Statistics()：统计考生信息，例如平均年龄、性别比例等。

**2.3.3 功能实现逻辑**

初始化 (Init\_linklist)：用户输入考生人数，依次输入每个考生的信息，构建链表。

插入 (Insert)：用户指定位置和考生信息，插入新节点到链表中。

删除 (Remove)：根据考号查找并删除对应的节点。

查找 (Research)：根据考号搜索链表，找到并显示相应考生信息。

修改 (Modify)：根据考号找到节点，修改节点中的考生信息。

统计 (Statistics)：遍历链表，进行如性别比例、平均年龄等统计分析。

打印 (Print)：遍历链表，打印出所有考生的信息。

这种设计提供了一套完整的功能集，可以有效地管理和操作考生信息。通过链表结构，实现了高效的数据处理，特别是在频繁的插入和删除操作中表现优异。

1. **项目实现**

**3.1 插入功能的实现**

**3.1.1 插入功能流程简介**

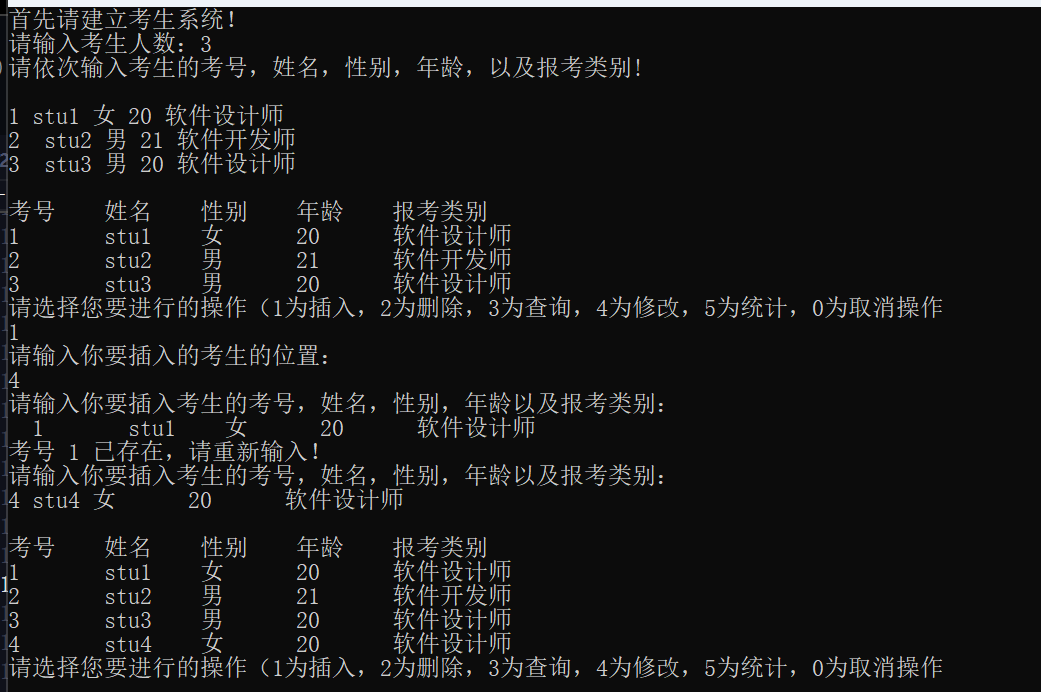
插入功能允许用户将新的考生信息添加到链表中。此功能首先要求用户输入考生的位置，然后根据输入的位置将新节点插入到链表中。如果输入的位置无效（即小于1或大于当前链表长度加1），系统将提示用户重新输入。此外，该功能还包括基本的输入验证和内存分配错误处理。

**3.1.2 插入功能核心代码**

以下是Insert方法的核心实现：

1. **bool** List::Insert()
2. {
3. cout << "请输入你要插入的考生的位置：" << endl;
4. **int** pos;
5. cin >> pos;
7. **if** (pos < 1 || pos > stu\_num + 1)
8. {
9. cout << "插入位置无效，请重新输入" << endl;
10. **return** **false**;
11. }
13. cout << "请输入你要插入考生的考号，姓名，性别，年龄以及报考类别" << endl;
14. link\_node\* pnew = **new** link\_node;
15. **if** (pnew == nullptr)
16. {
17. cout << "内存分配错误" << endl;
18. **return** **false**;
19. }
20. cin >> pnew->id >> pnew->name >> pnew->gender >> pnew->age >> pnew->category;
22. **if** (pos == 1)
23. {
24. pnew->next = first->next;
25. first->next = pnew;
26. }
27. **else**
28. {
29. link\_node\* current = first->next;
30. **for** (**int** i = 1; i < pos - 1; ++i)
31. {
32. current = current->next;
33. }
34. pnew->next = current->next;
35. current->next = pnew;
36. }
38. stu\_num++;
39. Print(first);
40. **return** **true**;
41. }

**3.1.3 插入功能错误测试示例截图**

****

在`List::Insert` 方法中，错误处理主要集中在两个方面：检查用户输入的插入位置的有效性以及确保新插入的考号不与现有的考号重复。以下是对这两个错误处理逻辑的具体描述：

1. 检查插入位置的有效性

当用户输入考生的插入位置时，程序需要验证这个位置是否有效。有效的插入位置应该是一个介于 1 和 `stu\_num + 1`（链表当前长度加 1）之间的整数。这个范围确保用户可以在链表的任何现有位置或末尾插入新节点。具体逻辑如下：

- 如果用户输入的位置小于 1 或大于 `stu\_num + 1`，程序将输出错误信息（"插入位置无效，请重新输入"），然后返回 `false` 并退出方法。

- 如果位置有效，程序继续执行后续的插入逻辑。

2. 确保考号不重复

在用户输入考生信息后，程序需要检查新考生的考号是否已在链表中存在。考号的唯一性是重要的，因为考号重复会导致数据的不准确。具体逻辑如下：

- 遍历链表，对每个节点的考号与新输入的考号进行比较。

- 如果发现链表中已存在相同的考号，程序将输出错误信息（"考号 [考号] 已存在，请重新输入！"），然后释放为新节点分配的内存，清空输入缓冲区，并开始新的输入循环，要求用户重新输入考生信息。

- 如果整个链表遍历完毕没有发现重复的考号，程序将继续执行插入操作。

通过以上两个错误处理逻辑，程序有效地确保了用户输入的插入位置合理，并保证了考号的唯一性，从而维护了链表数据的准确性和完整性。

**3.2 删除功能的实现**

**3.2.1 删除功能流程简介**

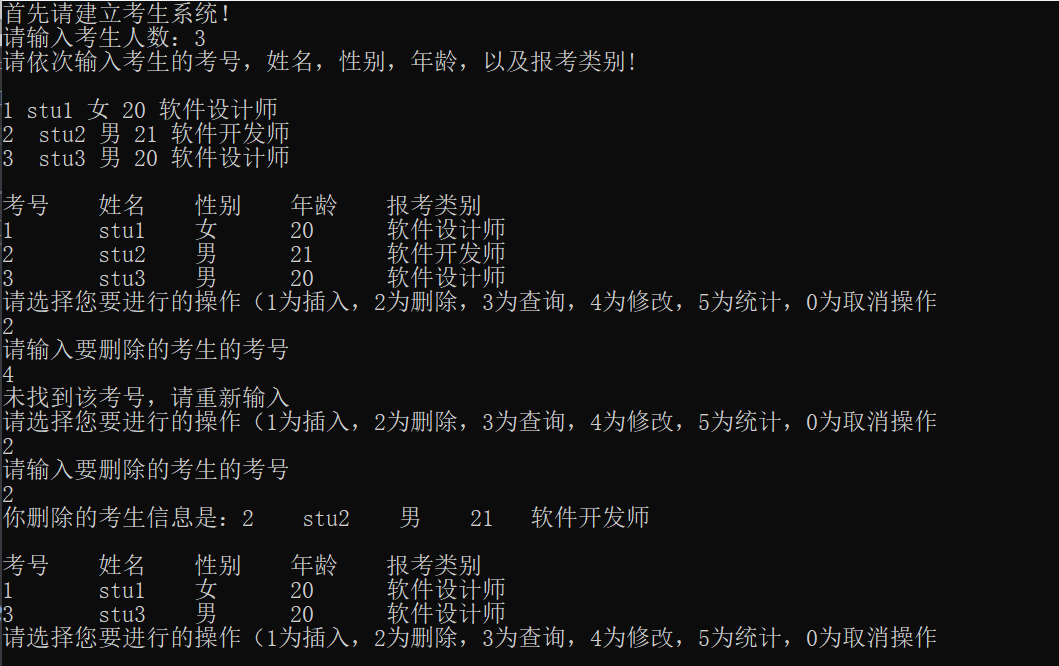
删除功能允许用户从链表中移除指定的考生信息。这一操作首先要求用户输入要删除的考生的考号。然后，程序遍历链表，寻找与输入的考号匹配的节点。如果找到匹配的考号，该节点将被删除。如果输入的考号不存在于链表中，系统将提示用户该考号未找到。此功能也包括基本的输入验证。

**3.2.2 删除功能核心代码**

以下是**Remove**方法的核心实现：

1. **bool** List::Remove()
2. {
3. cout << "请输入要删除的考生的考号：" << endl;
4. **int** stu\_id;
5. cin >> stu\_id;
7. link\_node\* current = first->next;
8. link\_node\* previous = first;
10. **while** (current != nullptr)
11. {
12. **if** (stu\_id == current->id)
13. {
14. cout << "你删除的考生信息是：" << current->id << "    " << current->name << "    " << current->gender << "    " << current->age << "   " << current->category << endl;
15. previous->next = current->next;
16. **delete** current;
17. stu\_num--;
18. Print(first);
19. **return** **true**;
20. }
21. previous = current;
22. current = current->next;
23. }
24. cout << "未找到该考号，请重新输入" << endl;
25. **return** **false**;
26. }

3.2.3 删除功能错误测试示例截图



在`List::Remove方法中，错误处理主要集中在以下方面：

1. 验证考号存在性：

- 遍历链表，比较每个节点的考号与用户输入的考号。

- 如果找到匹配的考号，删除对应的节点，并显示被删除的考生信息。

- 如果遍历完整个链表后未找到匹配的考号，输出错误信息（"未找到该考号，请重新输入"）。

2. 更新链表结构：

- 在找到要删除的节点后，调整链表，将被删除节点的前一个节点指向被删除节点的下一个节点。

- 释放被删除节点所占用的内存。

3. 维护考生数量：

- 每当成功删除一个节点时，更新链表中的考生数量统计(`stu\_num`)。

通过这种方法，程序能够有效地处理删除操作，确保链表的完整性和数据的准确性。

**3.3 查找功能的实现**

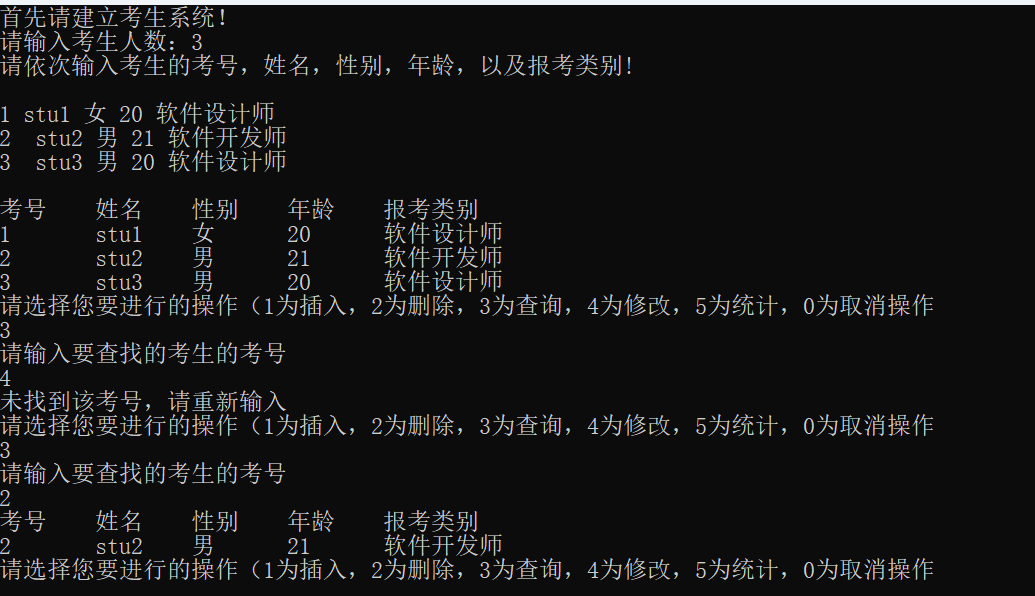
**3.3.1 查找功能流程简介**

查找功能使用户能够根据考号在链表中搜索特定的考生信息。用户输入想要查找的考生考号，程序随后遍历整个链表，寻找与输入考号匹配的节点。如果找到该考号对应的节点，程序将显示该考生的详细信息。如果未找到匹配的考号，系统将提示用户该考号不存在。此功能同样包括基本的输入验证。

**3.3.2 查找功能核心代码**

以下是**Research**方法的核心实现：

1. **bool** List::Research()
2. {
3. cout << "请输入要查找的考生的考号：" << endl;
4. **int** stu\_id;
5. cin >> stu\_id;
7. link\_node\* current = first->next;
9. **while** (current != nullptr)
10. {
11. **if** (stu\_id == current->id)
12. {
13. cout << "考号：" << current->id << ", 姓名：" << current->name << ", 性别：" << current->gender << ", 年龄：" << current->age << ", 报考类别：" << current->category << endl;
14. **return** **true**; // 找到匹配考号，返回true表示成功
15. }
16. current = current->next;
17. }
19. // 未找到匹配考号，给出友好提示
20. cout << "未找到该考号，请重新输入" << endl;
21. **return** **false**;
22. }
    * 1. 查找功能错误处理逻辑



在List::Research方法中，错误处理主要集中在确认考号的存在性方面：

验证考号存在性：

遍历链表，比较每个节点的考号与用户输入的考号。

如果找到匹配的考号，输出对应考生的详细信息。

如果遍历完整个链表后未找到匹配的考号，输出错误信息（"未找到该考号，请重新输入"）。

通过这种方法，程序能够有效地处理查找操作，确保用户可以快速找到所需的考生信息。查找功能不仅增加了系统的用户友好性，而且也提高了数据访问的效率。

**3.4 修改功能的实现**

**3.4.1 修改功能流程简介**

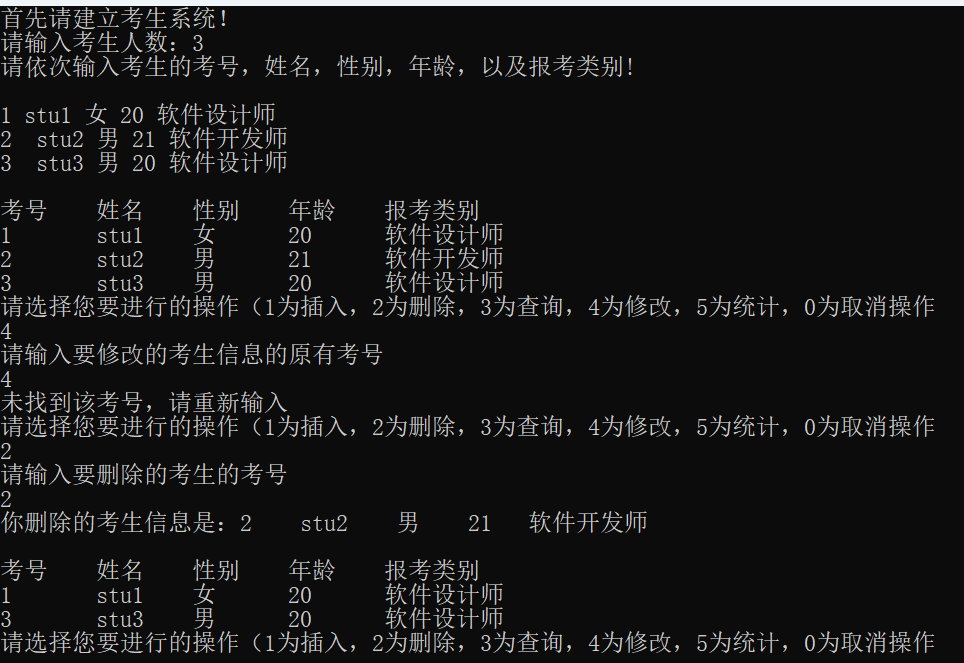
修改功能允许用户更新链表中特定考生的信息。此操作要求用户首先输入想要修改的考生的考号。然后程序遍历链表，寻找与输入的考号匹配的节点。如果找到该考号，程序允许用户输入新的考生信息以替换旧信息。如果未找到匹配的考号，系统将提示用户该考号不存在。此功能同样包含基本的输入验证。

**3.4.2 修改功能核心代码**

以下是Modify方法的核心实现：

1. **bool** List::Modify()
2. {
3. cout << "请输入要修改的考生信息的原有考号：" << endl;
4. **int** stu\_id;
5. cin >> stu\_id;
7. link\_node\* current = first->next;
9. **while** (current != nullptr)
10. {
11. **if** (stu\_id == current->id)
12. {
13. cout << "找到考生，当前信息为：" << endl;
14. cout << "考号：" << current->id << ", 姓名：" << current->name << ", 性别：" << current->gender << ", 年龄：" << current->age << ", 报考类别：" << current->category << endl;
15. cout << "请输入修改后该考生的考号，姓名，性别，年龄及报考类别：" << endl;
16. cin >> current->id >> current->name >> current->gender >> current->age >> current->category;
17. cout << "修改后的考生信息：" << endl;
18. cout << "考号：" << current->id << ", 姓名：" << current->name << ", 性别：" << current->gender << ", 年龄：" << current->age << ", 报考类别：" << current->category << endl;
19. **return** **true**; // 成功修改信息
20. }
21. current = current->next;
22. }
24. // 未找到匹配考号，给出友好提示
25. cout << "未找到该考号，请重新输入" << endl;
26. **return** **false**;
27. }

**3.4.3 修改功能错误处理逻辑**



在List::Modify方法中，错误处理主要集中在确认考号的存在性方面：

验证考号存在性：

遍历链表，比较每个节点的考号与用户输入的考号。

如果找到匹配的考号，输出当前考生的详细信息，并提示用户输入新的考生信息来进行更新。

更新节点信息后，显示新的考生信息以确认修改已经实施。

如果遍历完整个链表后未找到匹配的考号，输出错误信息（"未找到该考号，请重新输入"）。

通过这种方法，修改功能能够有效地允许用户更新考生信息，提高数据管理的灵活性，并确保用户能够容易地对链表中的数据进行更改。

3.5 统计功能的实现

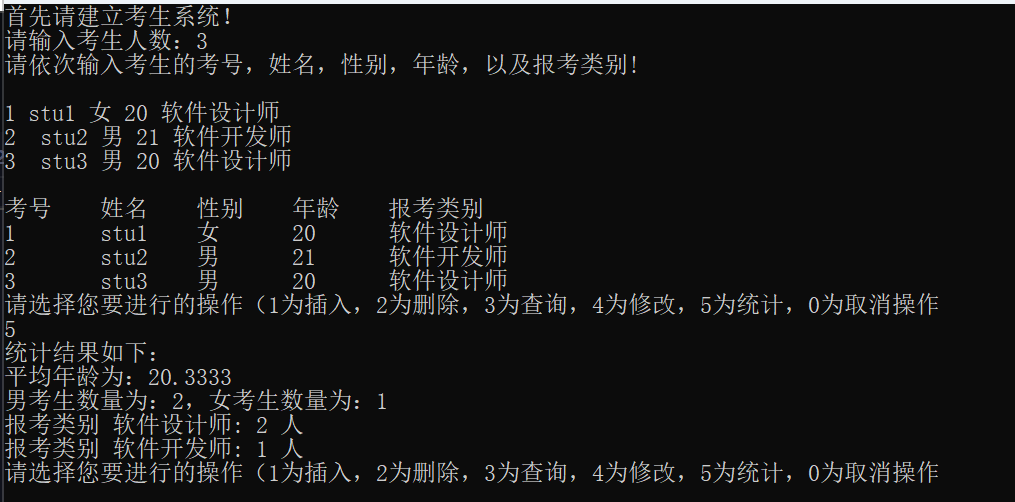
**3.5.1 统计功能流程简介**

统计功能使用户能够获取链表中存储的考生信息的概览。这一操作主要用于生成关于考生的各种统计数据，例如考生的平均年龄、性别比例以及不同报考类别的人数分布。此功能不需要用户输入，它会遍历整个链表并收集必要的信息来计算和显示统计结果。

**3.5.2 统计功能核心代码**

以下是Statistics方法的核心实现：

1. **bool** List::Statistics()
2. {
3. **if** (first->next == nullptr) // 检查链表是否为空
4. {
5. cout << "链表为空，无法进行统计。" << endl;
6. **return** **false**;
7. }
9. **int** total\_age = 0, male\_count = 0, female\_count = 0;
10. map<string, **int**> category\_count;
12. link\_node\* current = first->next;
13. **while** (current != nullptr)
14. {
15. total\_age += current->age;
16. **if** (strcmp(current->gender, "男") == 0)
17. male\_count++;
18. **else** **if** (strcmp(current->gender, "女") == 0)
19. female\_count++;
20. category\_count[current->category]++;
22. current = current->next;
23. }
25. **int** total\_students = male\_count + female\_count;
26. **double** average\_age = total\_students > 0 ? **static\_cast**<**double**>(total\_age) / total\_students : 0.0;
28. cout << "统计结果：" << endl;
29. cout << "总考生数：" << total\_students << endl;
30. cout << "平均年龄：" << average\_age << endl;
31. cout << "男性考生数：" << male\_count << ", 女性考生数：" << female\_count << endl;
32. **for** (auto& category : category\_count)
33. {
34. cout << "报考类别：" << category.first << ", 人数：" << category.second << endl;
35. }
37. **return** **true**;
38. }

3.5.3 统计功能错误处理逻辑

在List::Statistics方法中，主要的考虑是确保链表不为空，并且正确地收集和计算统计数据：

检查链表非空：

在开始统计之前，首先检查链表是否为空（即没有任何考生信息）。如果链表为空，输出提示信息，并返回false。

收集和计算统计数据：

遍历链表，收集考生的年龄、性别和报考类别。

计算总考生数、平均年龄、男性和女性考生的数量。

计算各报考类别的考生数量。

输出统计结果：

显示总考生数、平均年龄、性别比例。

对于每个报考类别，显示该类别的考生数量。

通过这种方法，统计功能能够为用户提供链表中考生数据的有用概览，增强了数据分析的能力，使得用户能够更好地理解和使用存储在链表中的信息。

**3.6 建立初始链表功能的实现**

**3.6.1 建立初始链表功能流程简介**

建立初始链表功能是用于在程序开始时创建链表，并填充初步的考生数据。该功能首先要求用户输入考生的数量，然后依次输入每个考生的信息，包括考号、姓名、性别、年龄和报考类别。输入的数据被用来构建链表的节点，从而形成初始的链表结构。这个功能为后续的增删改查操作提供了基础数据。

**3.6.2 建立初始链表功能核心代码**

以下是Init\_linklist方法的核心实现：

1. **bool** List::Init\_linklist(**int** n)
2. {
3. **if** (n <= 0)
4. {
5. cout << "人数不符合要求" << endl;
6. **return** **false**;
7. }
9. cout << "请依次输入考生的考号，姓名，性别，年龄，以及报考类别:" << endl;
10. stu\_num = n;
11. link\_node\* pcurrent = first;
13. **for** (**int** i = 0; i < n; i++)
14. {
15. link\_node\* pnew = **new** link\_node;
16. **if** (pnew == nullptr)
17. {
18. cout << "内存分配错误" << endl;
19. **return** **false**;
20. }
21. cin >> pnew->id >> pnew->name >> pnew->gender >> pnew->age >> pnew->category;
22. pnew->next = nullptr;
23. pcurrent->next = pnew;
24. pcurrent = pnew;
25. }
26. **return** **true**;
27. }

**3.6.3 建立初始链表功能错误处理逻辑**

在List::Init\_linklist方法中，主要的错误处理包括：

验证输入的考生数量：

首先检查用户输入的考生数量是否有效（大于0）。如果数量无效，输出错误信息，并返回false。

动态分配节点内存：

对于每个新考生，创建一个新的link\_node节点。如果内存分配失败（new link\_node返回nullptr），输出错误信息，并返回false。

收集每个考生的数据：

要求用户输入每个考生的详细信息，并将其存储在新创建的节点中。

构建链表结构：

将每个新创建的节点按顺序连接到链表的末端。

通过这种方法，Init\_linklist功能能够成功创建一个包含初步考生数据的链表，为链表的后续操作奠定基础。这个功能对于整个程序来说至关重要，因为它是数据输入和后续处理的起点。

1. **程序创新点**

**链表数据结构的应用：**

程序采用链表来存储考生信息，这是一种高效的数据结构，尤其适合于频繁的插入和删除操作。与使用数组或其他静态数据结构相比，链表提供了更高的灵活性和效率。

**用户友好的交互设计：**

程序提供了清晰的用户指引和提示，使得用户可以方便地进行操作，如插入、删除、查找、修改和统计。这种易于理解的交互方式可以提升用户体验，降低使用难度。

**综合性功能集成：**

程序不仅限于基本的增删查改操作，还包含了统计功能，这使得程序不仅是一个信息管理工具，也是一个数据分析工具。用户可以通过统计功能快速获得关于考生信息的洞见。

**面向实际应用场景的设计：**

程序的设计考虑了实际应用场景中的需求，如考生信息管理，使其具有实际应用价值。特别是在教育或考试管理领域，这种程序可能特别有用。

**灵活的数据修改与更新：**

修改功能允许用户不仅更改考生的基本信息，还能更新考号等关键数据。这提供了数据管理的高度灵活性，允许程序适应各种数据更新的需求。

1. **实验小结**

在本次实验中，我成功开发了一个考生信息管理系统，采用链表数据结构来有效地存储和处理考生数据。这个系统实现了基本的增删查改功能，包括创建初始链表、插入新节点、删除节点、查找节点、修改节点信息以及进行统计分析，满足了基础的数据管理需求。实验的核心在于运用链表结构，提高了数据操作的灵活性和效率，特别适合于频繁的插入和删除操作。同时，简洁明了的用户指令输入和反馈信息使程序易于操作和理解，而各种输入验证和错误处理则确保了数据的完整性和准确性。

过程中，我面临了一些挑战，如确保考号的唯一性，以及在删除和修改操作中正确处理链表节点的链接和断开，保证链表的完整性和一致性。这些挑战的克服不仅提升了我们的编程技能，特别是在C++和数据结构方面的能力，也锻炼了我的逻辑思维和问题解决能力。此外，实验强化了我对程序设计与实际应用场景和用户需求之间联系的理解。

未来，我计划进一步改进该系统，包括提高用户界面的友好性，引入更多高级功能如数据导出和模糊搜索，以及优化程序性能，特别是在处理大量数据方面。此外，考虑引入文件系统或数据库支持，实现数据的持久化存储，将进一步提升程序的实用性和可靠性。总的来说，这次实验不仅成功实现了目标，还为我们未来的学习和开发工作提供了宝贵的经验和启示。