****

**实验报告：两个有序链表序列的交集**

2351041

刘浩田

完成日期:2024.9.30

1. **索引**

### 概述

### 项目目的

### 项目环境

### 项目要求

### 项目实现

### 项目总结

1. **概述**

在现代计算机科学和软件开发中，数据结构扮演着至关重要的角色。链表，作为基础数据结构之一，因其独特的结构特点，在实现各种算法时提供了极大的灵活性。链表由一系列节点组成，每个节点包含数据部分和指向下一个节点的指针。与数组不同，链表中的元素不必在内存中连续存放，这使得它在插入和删除操作上具有显著优势。

本项目的核心任务是实现一个算法，用于找出两个非降序链表序列的交集，并将这些公共元素存储在一个新的链表中。非降序意味着链表中的元素按照从小到大的顺序排列，这为我们寻找交集提供了便利。在许多实际应用中，例如数据库操作、集合运算和网络通信等领域，寻找两个序列的交集是一个常见且重要的操作。

为了实现这一目标，我们首先需要定义链表的节点结构和链表本身的操作。节点结构通常包含数据字段和指向下一个节点的指针。链表操作则包括但不限于节点的插入、删除、查找和打印等。在本项目中，我们特别关注链表的查找操作，因为寻找交集本质上是一个查找问题。

设计算法时，我们需要考虑多种因素，包括算法的时间复杂度、空间复杂度以及如何高效地遍历和比较链表中的元素。一种直观的方法是使用两个指针分别遍历两个链表，比较它们的节点值，找到相同的元素后添加到新链表中。这种方法的时间复杂度为O(n\*m)，其中n和m分别为两个链表的长度。

在实现算法的过程中，我们还需要考虑各种边界条件，例如当一个链表为空时的情况，或者两个链表没有交集的情况。这些边界条件的处理对于确保算法的鲁棒性至关重要。

除了算法实现，我们还需要注意代码的可读性和可维护性。良好的代码风格、清晰的注释和合理的函数划分都是编写高质量代码的关键。此外，为了验证算法的正确性和稳定性，我们需要编写详尽的测试用例，覆盖各种可能的输入情况。

为了确保程序的广泛适用性，我们在设计时特别考虑了跨平台兼容性，使得该程序不仅在Windows操作系统上运行流畅，同时也兼容Linux环境，满足了不同用户群体的需求。通过使用标准C++语言和遵循Linux标准的函数，我们的程序能够在两大主流操作系统上提供一致的用户体验。

本项目不仅要求我们实现功能，还要求我们对代码进行优化，提高其运行效率。这就需要我们对算法进行深入的分析，找出可能的性能瓶颈，并采取相应的优化措施。优化过程本身就是一个不断试错和改进的过程，它需要我们具备细致的观察力和严谨的逻辑思维能力。

通过本项目，我们不仅能够提高对链表操作的熟练度，加深对算法逻辑的理解，还能够提升编程能力和问题解决能力。此外，项目还培养了我们的软件工程意识，包括需求分析、系统设计、测试和维护等各个阶段的实践。这些经验对于我们未来成为一名优秀的软件工程师具有重要的意义。

本项目是一个综合性的实践，涉及到数据结构的理解、算法的设计、代码的实现和测试等多个方面。通过这个项目，我们能够将理论知识应用到实际问题中，提升自己的技术能力和工程实践能力。

1. **项目目的**

本项目的核心目的在于通过设计和实现一个链表求交集的算法，深化对链表操作、算法逻辑以及程序设计的理解和应用能力。在计算机科学教育中，理论与实践相结合是培养解决实际问题能力的重要途径。通过本实验，不仅能够将课堂上学到的理论知识运用到具体项目中，还能够在实践中进一步提升自己的编程技巧和软件设计能力。

首先，实验旨在加强对链表数据结构的理解和应用。链表作为一种动态数据结构，在实现各种算法时提供了极大的灵活性。通过本实验，可以深入理解链表的特点和适用场景，掌握它们在实际问题中的应用方法。特别是在处理动态数据时，链表的灵活性和高效性显得尤为重要。

其次，实验强调了算法的重要性。一个优秀的算法能够显著提高程序的执行效率，对于链表求交集而言，这直接关系到程序的性能。例如，在实现链表交集查找功能时，选择合适的搜索算法可以加快检索速度；在进行数据比较时，合理的算法可以减少计算的复杂度。通过本实验，可以练习设计和优化算法，提高解决实际问题的效率。

再次，实验着重于提升的面向对象编程能力。面向对象编程是一种重要的编程范式，它通过将数据和操作封装成对象，提高了代码的可重用性和可维护性。在本实验中，需要构建多个类来表示不同的概念，如链表节点、链表本身等，并设计类的属性和方法来实现系统的功能。这要求不仅要掌握面向对象编程的语法，还要学会如何运用面向对象的思想进行系统设计。

此外，实验还旨在培养的软件工程意识。在实际的软件开发过程中，除了编写代码，还需要进行需求分析、系统设计、测试和维护等工作。通过本实验，可以体验软件开发的全过程，了解各个阶段的任务和要求。特别是在系统设计阶段，需要考虑系统的结构、模块划分、接口设计等问题，这对于提高的系统分析和设计能力具有重要意义。

最后，实验还关注于提升的用户体验意识。一个好的软件不仅要功能强大，还要易于使用。在本实验中，需要设计用户友好的交互界面，确保用户能够方便地进行操作。这要求不仅要关注程序的内部逻辑，还要考虑用户的需求和习惯，培养从用户的角度思考问题的能力。。

1. **项目环境**

## 系统：Windows11

## 编译器：VisualStudio2022x86

## 语言：C++

1. **项目要求**

本项目旨在开发一个考试报名系统，该系统能够满足高校考试报名工作的需求，提供高效的信息管理功能。以下是对项目的详细要求：

## 5.1功能要求

系统需要实现以下核心功能：

输入考生信息：能够通过用户输入或文件导入的方式添加考生信息。

输出考生信息：能够展示所有考生信息或特定考生的详细信息。

查询考生信息：能够根据准考证号、姓名或其他条件查询考生信息。

添加考生信息：在系统中添加新的考试信息，需要检查信息的有效性，如考号的唯一性。

修改考生信息：允许用户对已存在的考生信息进行修改。

删除考生信息：根据特定条件，如考号，删除考生信息。

## 5.2数据结构要求

数据组织：建议使用链表来存储考生信息，以支持高效的数据插入和删除操作。

数据完整性：确保所有考生信息的完整性和准确性，如检查必填项、格式等。

## 5.3用户界面要求

操作简便：用户界面应直观易用，操作流程简单明了。

输入验证：系统应提供输入验证，确保用户输入的数据符合要求。

错误处理：对于用户的错误操作，系统应提供清晰的错误提示。

## 5.4可复用性和可维护性

模块化设计：系统应采用模块化设计，便于后期维护和升级。

代码可读性：代码应具有良好的可读性，注释清晰，便于他人理解和修改。

## 5.5测试要求

测试用例：为每个功能编写详细的测试用例，确保功能的正确性和稳定性。

边界测试：进行边界条件测试，确保系统在极端情况下也能正常工作。

通过满足以上要求，考试报名系统将能够为用户提供一个功能全面、操作简便、安全可靠的考试报名管理工具，有效提升考试报名工作的效率和质量

1. **项目实现**

本项目实现一个用于计算两个有序链表序列交集的系统。以下是实现过程的详细描述：

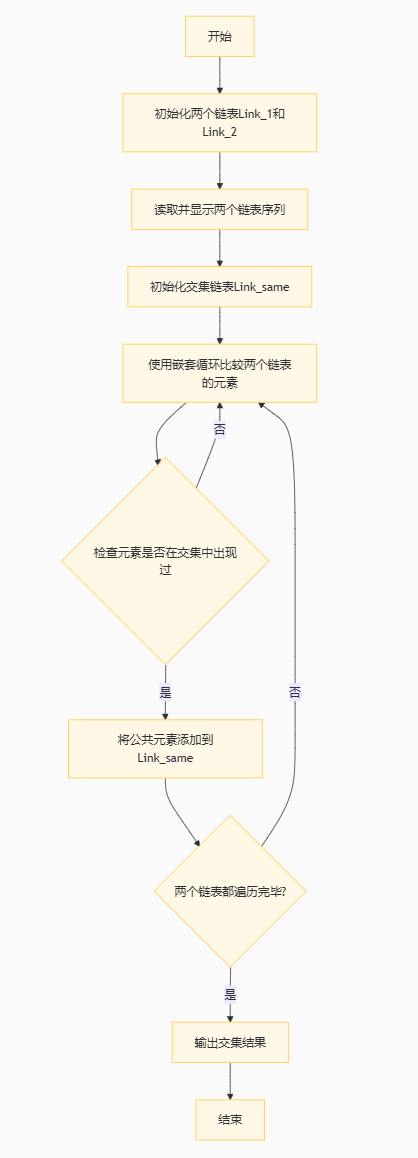


图 项目流程图

## 6.1 类设计

### ListNode类

目的：定义链表的节点。

方法：

构造函数：初始化节点的数据和指向下一个节点的指针。

### List类

目的：实现一个通用的链表，提供基本的链表操作。

方法：

构造函数：接受一个初始值列表，创建链表。

析构函数：释放链表所占用的内存。

Insert\_multiple：允许用户输入链表的多个值，以-1作为结束标志。

Find：查找特定值的节点。

Findpos：查找特定值的节点位置。

Truenum：返回链表中第n个节点的数据。

Insert：在链表的指定位置插入新的节点。

Remove：移除链表中指定位置的节点。

Print：打印链表中的所有元素。

single\_Print：打印链表中的单个节点。

Length：返回链表的长度。

## 6.2 算法实现

6.2.1输入处理  
详细步骤：

提示用户输入两个非降序链表序列。

循环接收每个序列的节点数据，直到输入-1表示序列结束。

将输入的数据构建成链表。

|  |
| --- |
| 附件、输入处理函数 |
| void Insert\_multiple()  {  int last = 0;  for (int k = 1;; k++) {  int getnum;  while (1) {  cin >> getnum;  if (cin.fail()) {  cin.clear();  cin.ignore(IGNORE\_SIZE, '\n');  cout << "输入非正整数，请重新输入" << endl;  continue;  } else if (getnum < 0 && getnum != -1) {  cin.ignore(IGNORE\_SIZE, '\n');  cout << "输入非正整数，请重新输入" << endl;  continue;  } else if (getnum < last && getnum != -1) {  cin.ignore(IGNORE\_SIZE, '\n');  cout << "输入了降序链表，请在输入错误位置重新输入" << endl;  continue;  } else {  break;  }  }  if (getnum == -1) {  break;  }  last = getnum;  Insert(getnum, k);  }  } |

6.2.2交集计算  
详细步骤：

初始化两个指针，分别指向两个链表的头节点。

比较两个指针所指向的节点的数据，取较小的值进行下一步比较。

如果两个节点的值相等，则将该值添加到交集中。

移动指针到下一个节点，继续比较，直到其中一个链表结束。

如果一个链表结束，则将另一个链表剩余的节点值添加到交集中（如果存在重复则只添加一次）。

|  |
| --- |
| 附件、交集计算部分函数 |
| int pos = 1;  int check = 0;  List<int> Link\_same = {};  for (int i = 1; i <= Link\_1.Length(); i++) {  for (int j = 1; j <= Link\_2.Length(); j++) {  if (Link\_1.Truenum(i) == Link\_2.Truenum(j)) {  if (!Link\_same.Findpos(Link\_2.Truenum(j))) {  Link\_same.Insert(Link\_1.Truenum(i), pos);  check++;  pos++;  }  }  }  } |

6.2.3输出结果  
详细步骤：

遍历交集中的链表，打印每个节点的数据。

如果交集为空，则输出"NULL"，最后输出查找具体情况。

|  |
| --- |
| 附件、交集计算部分函数 |
| cout << "对比两链表结果如下：" << endl;  cout << "相同数据为：";  if (!Link\_same.Length()) {  cout << "NULL" << endl;  } else {  Link\_same.Print();  cout << endl;  }  cout << "具体说明为：";  if (!Link\_1.Length() || !Link\_2.Length()) {  cout << "其中一个序列为空的情况" << endl;  } else if (!Link\_same.Length()) {  cout << "交集为空的情况" << endl;  } else if (check == Link\_1.Length() && check == Link\_2.Length()) {  cout << "完全相交的情况" << endl;  } else if (check == Link\_1.Length() || check == Link\_2.Length()) {  cout << "其中一个序列完全属于交集的情况" << endl;  } else {  cout << "一般情况" << endl;  } |

**6.3 主函数设计**

1、初始化两个链表Link\_1和Link\_2。

2、分别调用Insert\_multiple方法读取两个链表的序列。

3、使用嵌套循环比较两个链表的元素，调用Truenum方法获取指定位置的元素，通过Findpos检查元素是否在交集中出现过。

4、将找到的公共元素使用Insert方法添加到Link\_same链表中。

5、根据Link\_same的长度，调用Print方法输出交集结果或输出"NULL"。

|  |
| --- |
| 附件、主函数部分函数 |
| int main()  {  cout << "请输入第一行非降序链表序列，以-1结尾" << endl;  List<int> Link\_1 = {};  Link\_1.Insert\_multiple();  cout << "你输入的第一行非降序链表序列为：" << endl;  if (!Link\_1.Length()) {  cout << "NULL" << endl;  } else {  Link\_1.Print();  cout << endl;  }  cin.ignore(IGNORE\_SIZE, '\n');  cout << "请输入第二行非降序链表序列，以-1结尾" << endl;  List<int> Link\_2 = {};  Link\_2.Insert\_multiple();  cout << "你输入的第二行非降序链表序列为：" << endl;  if (!Link\_2.Length()) {  cout << "NULL" << endl;  } else {  Link\_2.Print();  cout << endl;  }  int pos = 1;  int check = 0;  List<int> Link\_same = {};  寻找交集部分  输出部分  return 0;  } |

**6.4 测试用例设计**

**正常情况**：两个链表序列均有多个相同的元素。

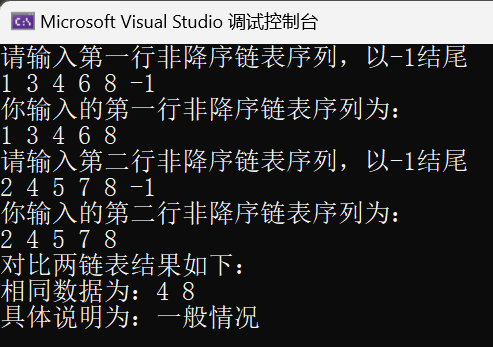
输入：1 3 4 6 8 -1

2 4 5 7 8 -1

预期输出：4 8

一般情况

实验结果：



**交集为空**：两个链表序列无交集。

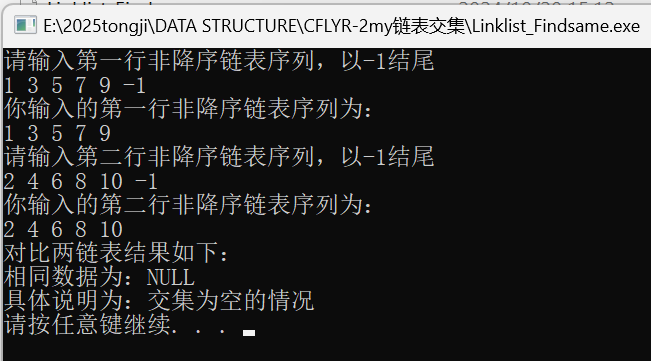
输入：1 3 5 7 9 -1

2 4 6 8 10 -1

预期输出：NULL

交集为空情况

实验结果：



**一个序列为空**：一个输入序列为空。

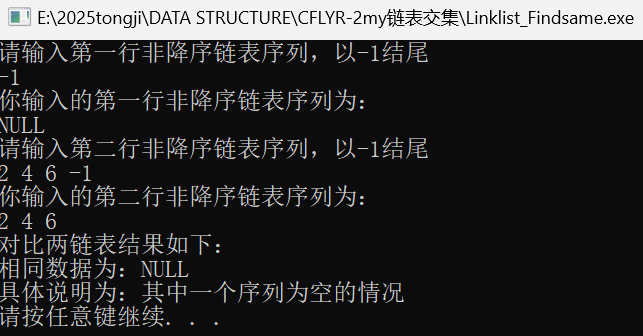
输入： -1

2 4 6 -1

预期输出：NULL

其中一个序列为空的情况

实验结果：



**完全相交的情况**：两个序列完全相同

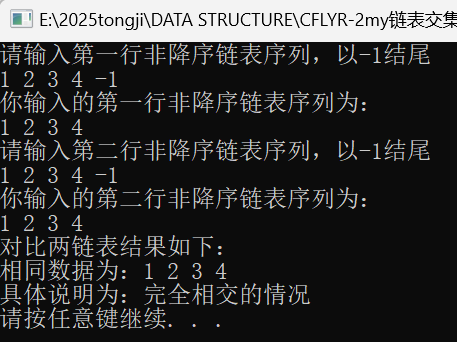
输入： 1 2 3 4 -1

1 2 3 4 -1

预期输出：1 2 3 4

完全相交的情况

实验结果：



**全属于交集**：一个链表序列完全被另一个包含

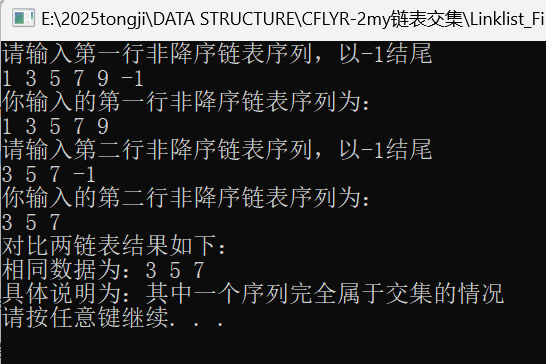
输入：1 3 5 7 9 -1

3 5 7 -1

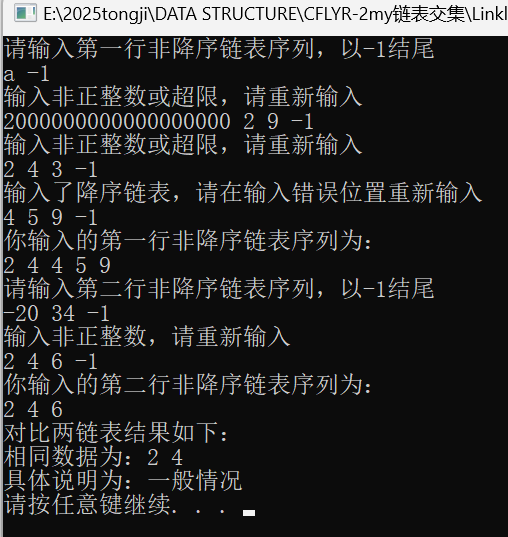
预期输出：3 5 7

其中一个序列完全属于交集的情况

实验结果：



**错误输入：输入非法字符或降序序列**



通过满足以上实现要求，本项目将能够为用户提供一个功能全面、操作简便、安全可靠的链表序列交集计算工具，有效提升处理链表相关任务的效率和质量。

1. **项目总结**

通过本次实现两个有序链表序列交集的系统设计和开发，我获得了宝贵的实践经验，并在多个方面得到了显著的提升。以下是对本次项目的总结：

## 7.1 学习体会

本次实验使我深刻体会到了数据结构和算法理论知识在解决实际问题中的应用价值。

通过使用面向对象的方法设计链表和节点类，我加深了对面向对象设计原则的理解，并提高了代码的可读性和可维护性。

从需求分析到系统设计，再到实现和测试，我体验了完整的软件开发流程，这对我的系统分析和设计能力的提升具有重要意义。

## 7.2 技术收获

在项目中，我提高了我的C++编程技能，特别是在类模板的使用、内存管理和输入输出操作等方面。

我深入理解了链表的工作原理和操作，特别是在实现动态数据结构时，链表的灵活性和高效性显得尤为重要。

在实现链表交集算法的过程中，我锻炼了逻辑思考和问题解决的能力，学习了如何设计和优化算法来提高程序的效率。

## 7.3 问题与挑战

在用户输入不符合预期时，如何保证系统的健壮性是一个难题。我通过增加输入验证和异常捕获机制来提高系统的稳定性。

设计一个简洁明了的用户界面也是一个挑战。我通过不断迭代和改进用户界面，使其更加直观和易用。

## 7.4 改进方向

功能扩展：未来可以增加更多功能，如链表的可视化展示，或者增加其他数据结构的交集查找功能。

性能提升：可以探索更高效的数据结构，如使用哈希表来存储链表元素，以进一步提高查找效率。

用户体验优化：可以进一步优化用户界面和交互设计，提供更加友好和直观的用户体验。

## 7.5 结论

这次课程设计是一次宝贵的学习经历。它不仅让我将所学知识应用于实践，还让我学会了如何分析和解决实际问题。通过独立完成项目，我克服了各种挑战，最终完成了一个功能完善、性能良好的链表求交集系统。我相信，这次经历将对我未来的学习和工作产生积极的影响。