



同濟大學  
TONGJI UNIVERSITY

## 数据结构课程设计

### 项目说明文档

### 有序链表交集

姓 名： 马小龙

学 号： 2353814

学 院： 计算机科学与技术学院（软件学院）

专 业： 软件工程

指导教师： 张颖

二〇二三年十月二十九日

# 目录

第 1 章 项目分析.....	1
1.1 项目背景分析.....	1
1.2 项目需求分析.....	1
1.3 项目功能分析.....	2
1.3.1 有序链表输入功能.....	2
1.3.2 有序链表输出功能.....	2
1.3.3 有序链表处理功能.....	2
1.3.4 异常处理功能.....	2
第 2 章 项目设计.....	3
2.1 数据结构设计.....	3
2.2 结构体与类设计.....	3
2.2.1 链表结点 (Node) 设计.....	3
2.2.1.1 概述.....	3
2.2.1.2 结构体定义.....	3
2.2.1.3 数据成员.....	4
2.2.1.4 构造函数.....	4
2.2.1.5 成员函数.....	4
2.2.2 链表 (LinkedList) 设计.....	4
2.2.2.1 概述.....	4
2.2.2.2 LinkedList 类定义.....	4
2.2.2.3 私有数据成员.....	5
2.2.2.4 构造函数与析构函数.....	5
2.2.2.5 公有成员函数及类.....	6
2.2.3 迭代器设计.....	7
2.2.3.1 概述.....	7
2.2.3.2 迭代器类定义.....	7
2.2.3.3 私有数据成员.....	7
2.2.3.4 构造函数.....	7
2.2.3.5 公有成员函数.....	8
2.3 项目框架设计.....	8
2.3.1 项目框架流程图.....	8
2.3.2 项目框架流程.....	8
第 3 章 项目功能实现.....	9
3.1 项目主体架构.....	9
3.1.1 实现思路.....	9

3.1.2 流程图.....	9
3.1.3 核心代码.....	9
3.1.4 示例.....	10
3.2 有序链表输入功能.....	10
3.2.1 实现思路.....	10
3.2.2 流程图.....	11
3.2.3 核心代码.....	11
3.2.4 示例.....	12
3.3 有序链表输出功能.....	12
3.3.1 实现思路.....	12
3.3.2 流程图.....	13
3.3.3 核心代码.....	13
3.3.4 示例.....	14
3.4 有序链表处理功能.....	14
3.4.1 实现思路.....	14
3.4.2 流程图.....	14
3.4.3 核心代码.....	15
3.4.4 示例.....	16
3.5 异常处理功能.....	16
3.9.1 输入非法的异常处理.....	16
3.9.2 动态内存申请失败的异常处理.....	16
第4章 项目测试.....	18
4.1 考生信息插入功能测试.....	18
4.2 考生信息删除功能测试.....	18
4.3 考生信息查找功能测试.....	19
第5章 相关说明.....	21
5.1 编程语言.....	21
5.2 Windows 环境.....	21
5.3 Linux 环境.....	21

## 第 1 章 项目分析

### 1.1 项目背景分析

在数据结构与算法的应用中，链表是一种常见的线性数据结构，广泛应用于各类信息处理场景。在实际开发中，需求往往涉及到对两个有序链表的交集进行求解，这不仅是一个基本的算法问题，也是多个实际应用场景中的核心需求之一。有效求解有序链表的交集能够帮助用户快速获取所需的信息，提升数据处理的效率。

随着数据量的不断增加，传统的线性搜索方式在处理链表时显得尤为低效。这种情况下，采用高效的算法来求解有序链表的交集变得尤为重要。通过设计优化的算法，我们可以在时间和空间复杂度上达到更好的平衡，从而实现更快速、准确的数据检索。这一过程不仅能够减少计算资源的消耗，还能显著提升用户体验。

在信息化迅猛发展的背景下，开发一个能够高效求解两个有序链表序列交集的系统，将为软件开发、数据分析等领域提供强有力的技术支持。这种系统的实现，不仅能促进信息的高效整合，还能够为企业或组织提供更加精确的数据决策依据。因此，深入研究和开发此类算法具有重要的理论意义和广泛的应用前景，为数据处理的现代化提供了坚实的基础。

### 1.2 项目需求分析

本项目需要实现一个能够快速求解两个有序链表交集的系统，提供准确、高效的数据处理能力，支持用户进行相关的数据分析。

系统应允许用户输入两个有序链表，支持常见的数据格式，例如数组或链表结点。在处理方面，系统需要实现一种高效的交集求解算法，以确保在  $O(n + m)$  的时间复杂度内完成计算。

在技术需求方面，系统将使用 C++ 等主流编程语言进行开发，并采用适合的线性数据结构（链表），以确保数据存储与处理的高效性。此外，系统还需具备错误处理能力，以应对无效输入（例如空链表或非有序链表），并提供相应的错误提示。

系统应当兼容 Windows、Linux 等主流操作系统，确保广泛应用。

## 1.3 项目功能分析

本项目旨在开发一套高效的求解有序链表的交集的系统。系统支持输入两个有序链表、求解链表交集、输出得的交集结果，，并显示交集元素的数量，以及进行异常处理，从而实现有序链表的交集的求解。以下是项目主要功能的详细信息。

### 1.3.1 有序链表输入功能

允许用户输入两个有序链表，输入时，程序会验证考生的基本信息是否符合规范，如不规范，需要重新输入。具体规范要求如下：

1. 用户输入的数据类型应当为正整数；
2. 每个链表输入的数据必须为非降序序列，并以-1 表示序列的结尾；
3. 数字应当使用空格间隔。

### 1.3.2 有序链表输出功能

系统应能将求得的交集结果以清晰的格式展示给用户，需要将交集结果打印到控制台，同时显示交集元素的数量，提供简单的统计信息。

### 1.3.3 有序链表处理功能

核心功能是实现高效的交集求解算法。该算法应能够在  $O(n + m)$  的时间复杂度内完成计算，其中  $n$  和  $m$  分别为两个链表的长度。具体包括：遍历两个链表，比较当前元素，找到共同元素；将交集结果存储在合适的数据结构中，以备后续输出。

### 1.3.4 异常处理功能

系统需具备健壮的错误处理能力，能够识别和处理各种潜在错误情况

## 第2章 项目设计

### 2.1 数据结构设计

本项目设计使用链表作为主要数据结构，用来储存有序序列的数据，主要是基于以下方面：

1. 链表的动态大小特性使得在处理不确定数量的元素时非常灵活。在实际应用中，两个链表的长度可能不同，链表可以根据实际需要动态调整，避免了固定大小数组可能带来的空间浪费。

2. 从系统维护和扩展的复杂度考虑，链表提供了更好的可扩展性和易于修改的特性。在未来需要进行功能扩展时，链表结构的灵活性将大大简化设计和实现过程。

### 2.2 结构体与类设计

为了使链表更加通用，本链表设计为链表模板设计，为链表添加了许多实用性功能。

#### 2.2.1 链表结点（Node）设计

##### 2.2.1.1 概述

Node 用于储存信息，包括结点储存的具体信息内容和下一结点的地址。

##### 2.2.1.2 结构体定义

```
struct Node {  
    T data;  
    Node<T>* next;  
    operator T();  
    operator T& ();  
    operator const T& () const;  
    Node<T>& operator=(const Node<T>& node);  
    bool operator==(const Node<T>& node) const;  
    Node(Node<T>* ptr = nullptr);  
    Node(const T& item, Node<T>* ptr = nullptr);  
};
```

```
};
```

### 2.2.1.3 数据成员

T data: 数据域, 存储结点的数据

Node<T>\* next: 指针域, 指向下一个结点的指针

### 2.2.2.4 构造函数

```
Node(Node<T>* ptr = nullptr);
```

构造函数, 初始化指针域。

```
Node(const T& item, Node<T>* ptr = nullptr);
```

构造函数, 初始化数据域和指针域。

### 2.2.1.5 成员函数

```
operator T();
```

允许 Node 对象被视为其数据类型的一个副本。

```
operator T& ();
```

允许 Node 对象被视为其数据类型的一个引用。

```
operator const T& () const;
```

提供了对数据的常量引用访问, 确保在只读上下文中使用。

```
Node<T>& operator=(const Node<T>& node);
```

=运算符重载, 允许一个 Node 对象通过赋值操作符=从另一个 Node 对象复制数据。

```
bool operator==(const Node<T>& node) const;
```

==运算符重载, 用于比较两个 Node 对象是否相等;

## 2.2.2 链表 (LinkedList) 设计

### 2.2.2.1 概述

该通用模板类 LinkedList 用于表示单链表。此链表头结点只做定位用途, 不储存数据, 头结点的下一结点为数据储存的起点。链表结点由 Node 结构体表示, 其中包含数据 和指向下一个结点的指针。该链表提供了一系列基本操作函数, 包括结点的插入、删除、查找、访问等, 以及链表的构造和析构, 满足了常见的链表操作需求。

### 2.2.2.2 LinkedList 类定义

```
template <typename T>
class LinkedList {
```

```

public:
    class iterator;
    iterator begin();
    iterator end();
    LinkList();
    LinkList(const T& x);
    LinkList(const LinkList<T>& L);
    ~LinkList();
    void makeEmpty();
    int Length();
    Node<T>* getHead();
    Node<T>* Search(T x);
    Node<T>* Locate(int i);
    bool getData(int i, T& x);
    bool setData(int i, const T& x);
    bool Insert(int i, const T& x);
    bool Remove(int i, T& x);
    bool IsEmpty() ;
    static bool IsFull() ;
    void inputFront(const T& x);
    void inputRear(const T& x);
    void output();
    LinkList<T>& operator=(const LinkList<T>& other);
    T& operator[](int index);
private:
    Node<T>* head;
};

```

### 2.2.2.3 私有数据成员

Node<T>\* head: 指向链表头结点的指针

### 2.2.2.4 构造函数与析构函数

LinkList();

默认构造函数，创建一个空链表。

LinkList(const T& x);



转换构造函数，创建一个包含头结点和一个数据结点的链表。

`LinkedList(const LinkedList<T>& L);`

复制构造函数，通过复制另一个链表创建新链表。

`~LinkedList();`

析构函数，释放链表的内存资源，包括所有结点的内存。

### 2.2.2.5 公有成员及数据类型

`class iterator;`

迭代器类定义。

`iterator begin();`

迭代器初始结点，指向链表头结点后一个结点

`iterator end();`

迭代器末结点，链表尾后一个结点，及 `nullptr`。

`void makeEmpty();`

清空链表，释放除头结点所有结点的内存。

`int Length();`

获取链表中结点的个数。

`Node<T>* getHead();`

获取链表头结点的指针。

`Node<T>* Search(T x);`

搜索链表中值为 `x` 的结点，返回该结点的指针，若不存在返回 `nullptr`。

`Node<T>* Locate(int i);`

返回链表中第 `i` 个结点的指针，若 `i` 超出链表长度或小于 0，则返回 `nullptr`

`bool getData(int i, T& x);`

获取链表中第 `i` 个结点的数据，并通过引用返回。返回值为操作是否成功

`bool setData(int i, const T& x);`

设置链表中第 `i` 个结点的数据。返回值为操作是否成功

`bool Insert(int i, const T& x);`

在链表中第 `i-1` 个结点后插入新结点，成为第 `i` 的结点（头结点记为第零个结点）。返回值为操作是否成功。

`bool Remove(int i, T& x);`

删除链表中第 `i` 个结点，并通过引用返回其数据。返回值为操作是否成功。

`bool IsEmpty();`

判断链表是否为空（只有头结点即为空）

```
void inputFront(const T& x);
```

在链表的开头（head 之后）插入一个新元素

```
void inputRear(const T& x);
```

在链表的尾部插入一个新元素。

```
void output();
```

输出链表中所有结点的数据。

```
LinkList<T>& operator=(const LinkList<T>& other);
```

赋值运算符重载，将一个链表的所有值赋给另外一个，形成两个相同的链表

```
T& operator[](int index);
```

下标运算符重载，返回第 index 结点的数据

## 2.2.3 迭代器设计

### 2.2.3.1 概述

求交集需要遍历访问链表中的所有结点，为了实现这一过程，有三种方法：1. 采用结点地址→data→number 的方式访问，通过结点地址→next 转到下一结点，这样太过于麻烦，且模块化不够好；2. 采用下标 [] 运算，但每一次都要从头访问，时间复杂度较高。最好的方式是采用迭代器。

### 2.2.3.2 迭代器类定义

```
class iterator {
private:
    Node<T>* current;
public:
    iterator(Node<T>* ptr = nullptr);
    iterator& operator++();
    T& operator*();
    bool operator==(const iterator& other) const;
    bool operator!=(const iterator& other) const ;
};
```

### 2.2.3.3 私有数据成员

Node<T>\* head: 指向当前结点的指针

### 2.2.3.5 构造函数

```
iterator(Node<T>* ptr = nullptr);
```

构造函数，设定当前结点地址。

### 2.2.3.5 公有成员函数

`iterator& operator++();`

重载前缀递增运算符。

`T& operator*();`

重载解引用运算符，返回结点对象。

`bool operator==(const iterator& other) const;`

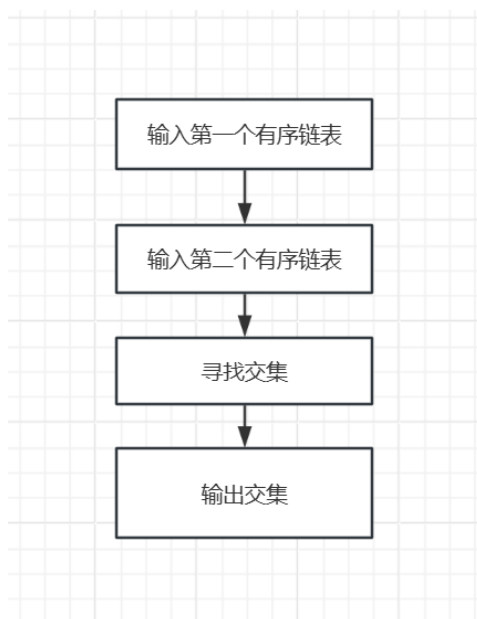
重载等于运算符，判断结点对象是否相等。

`bool operator!=(const iterator& other) const;`

重载不等于运算符，判断结点对象是否不相等。

## 2.3 项目框架设计

### 2.3.1 项目框架流程图



### 2.3.2 项目框架流程

1. 输入第一个有序链表；
2. 输入第二个有序链表；
3. 寻找有序链表的交集，将其放在一个新的链表；
3. 输出两个有序链表的交集，以及简单统计信息。

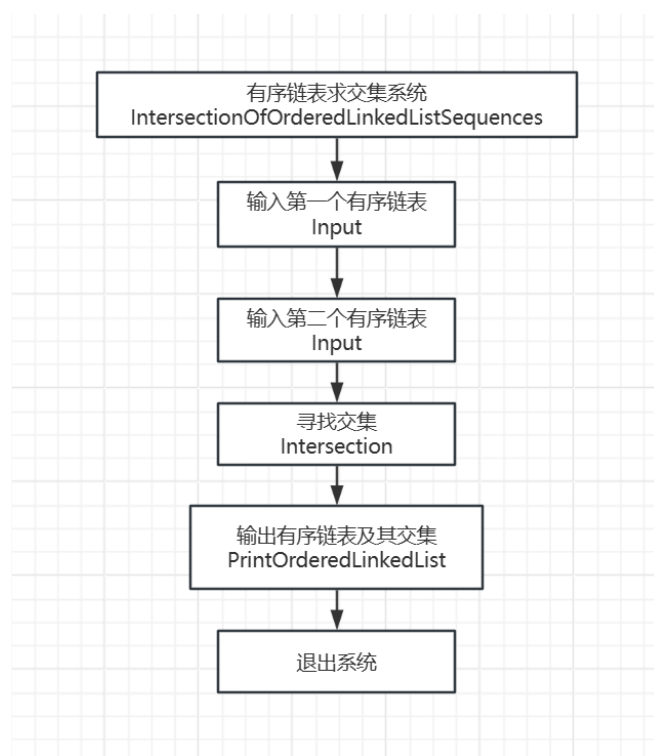
## 第3章 项目功能实现

### 3.1 项目主体架构

#### 3.1.1 实现思路思路

1. 进入 IntersectionOfOrderedLinkedListSequences 函数以进入有序链表求交集系统。
2. 调用 Input 函数依次输入第一和第二个非降序链表；
3. 调用 Intersection 函数求有序链表的交集；
4. 输出两个有序链表的交集；
5. 退出有序链表求交集系统。

#### 3.1.2 流程图



#### 3.1.3 核心代码

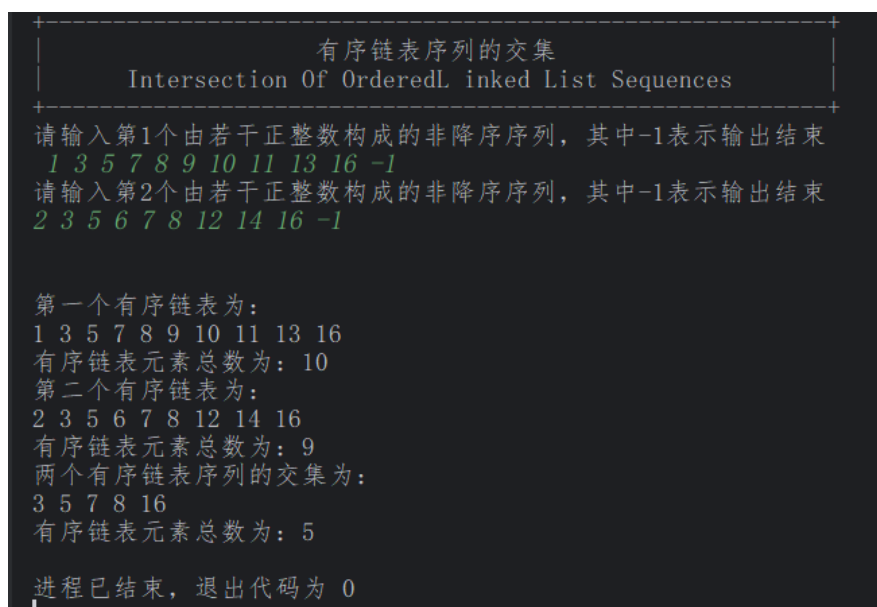
```
void IntersectionOfOrderedLinkedListSequences()
{
```

```

std::cout << "+-----+\n";
std::cout << "|                                有序链表序列的交集
\n";
std::cout << "|                Intersection Of OrderedL inked List Sequences
\n";
std::cout << "+-----+\n";
LinkedList<int> first = Input(1);
LinkedList<int> second = Input(2);
LinkedList<int> intersection = Intersection(first, second);
std::cout<<"\n\n";
PrintOrderedList(first,"第一个有序链表为: \n");
PrintOrderedList(second,"第二个有序链表为: \n");
PrintOrderedList(intersection,"两个有序链表序列的交集为: \n");
}

```

### 3.1.4 示例



```

+-----+
|                                有序链表序列的交集
|                Intersection Of OrderedL inked List Sequences
+-----+
请输入第1个由若干正整数构成的非降序序列，其中-1表示输出结束
1 3 5 7 8 9 10 11 13 16 -1
请输入第2个由若干正整数构成的非降序序列，其中-1表示输出结束
2 3 5 6 7 8 12 14 16 -1

第一个有序链表为：
1 3 5 7 8 9 10 11 13 16
有序链表元素总数为：10
第二个有序链表为：
2 3 5 6 7 8 12 14 16
有序链表元素总数为：9
两个有序链表序列的交集为：
3 5 7 8 16
有序链表元素总数为：5

进程已结束，退出代码为 0

```

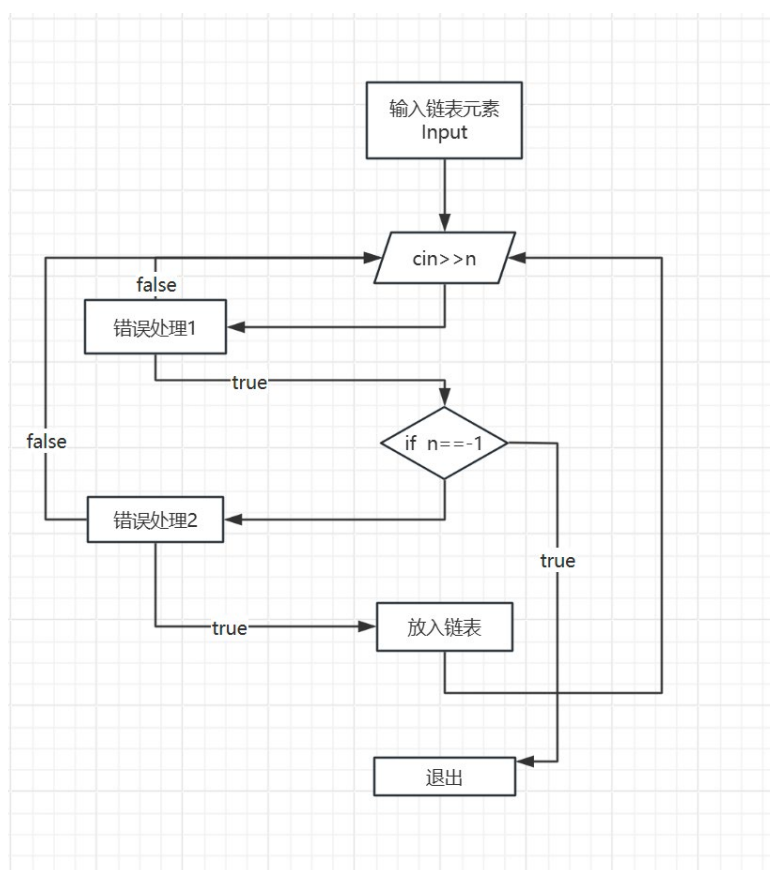
## 3.2 有序链表输入功能

### 3.2.1 实现思路

考生信息录入功能的函数名为 Input，考生信息录入功能实现的思路为：

1. IntersectionOfOrderedLinkedListSequences 通过调用两次 Input 函数，并传递参数；
2. 进入 Input 函数，首先提示输入信息（根据参数不同有所差异）；
3. 进入一个无限循环开始逐个输入有序链表元素，若链表元素输入有一定规范，若不符合规范需要重新输入当前元素；
4. 将输入的元素链接到链表末尾；
5. 当输入为-1时，直接退出循环及 Input 函数，开始下一个链表输入或进行其他操作。

### 3.2.2 流程图



### 3.2.3 核心代码

```

LinkedList<int> Input(int rank)
{
    std::cout << "请输入第" << rank << "个由若干正整数构成的非降序序
列，其中-1 表示输出结束\n";
    LinkedList<int> temp;

```

```
double temp_element, previous = 1;
while (true) {
    std::cin >> temp_element;
    int element = static_cast<int>(temp_element);
    if (std::cin.fail() || temp_element < -1 || temp_element == 0 ||
temp_element != element) {
        std::cout << "输入的元素包含非正整数项，请重新输入\n";
        ClearBuffer();
        continue;
    }
    if (element == -1) {
        ClearBuffer();
        break;
    }
    if (element < previous) {
        std::cout << "输入的元素中存在前项大于后项的情况，请重新输入\n";
        ClearBuffer();
        continue;
    }
    temp.inputRear(element);
    previous = element;
}
return temp;
}
```

#### 3.2.4 示例

示例同 3.1.4

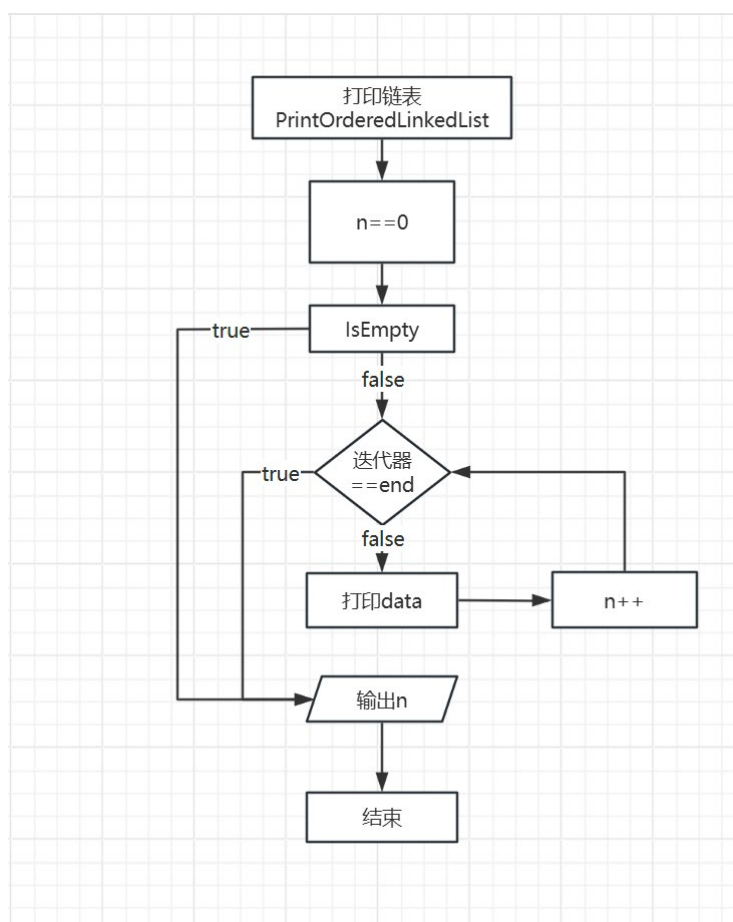
### 3.3 有序链表输出功能

#### 3.3.1 实现思路

有序链表功能的函数名为 PrintOrderedLinkedList，输出考有序链表功能实现的思路为：

1. 打印输出提示信息；
2. 检查链表是否为空，若为空则输出“NULL”；
3. 通过迭代器的方式输出链表每个结点的信息；
4. 输出链表元素总数；

### 3.3.2 流程图



### 3.3.3 核心代码

```

void PrintOrderedLinkedList(LinkList<int>& link, const char* str)
{
    std::cout<<str;
    bool first=true;
    int n=0;

```



```
if(link.Length()==0)
    std::cout<<"NULL";
else
    for(auto node:link) {
        if(!first)
            std::cout<<" ";
        std::cout<<node;
        first=false;
        n++;
    }
    std::cout<<"有序链表元素总数为: "<<n<<"\n";
}
```

### 3.3.4 示例

示例同 3.1.4

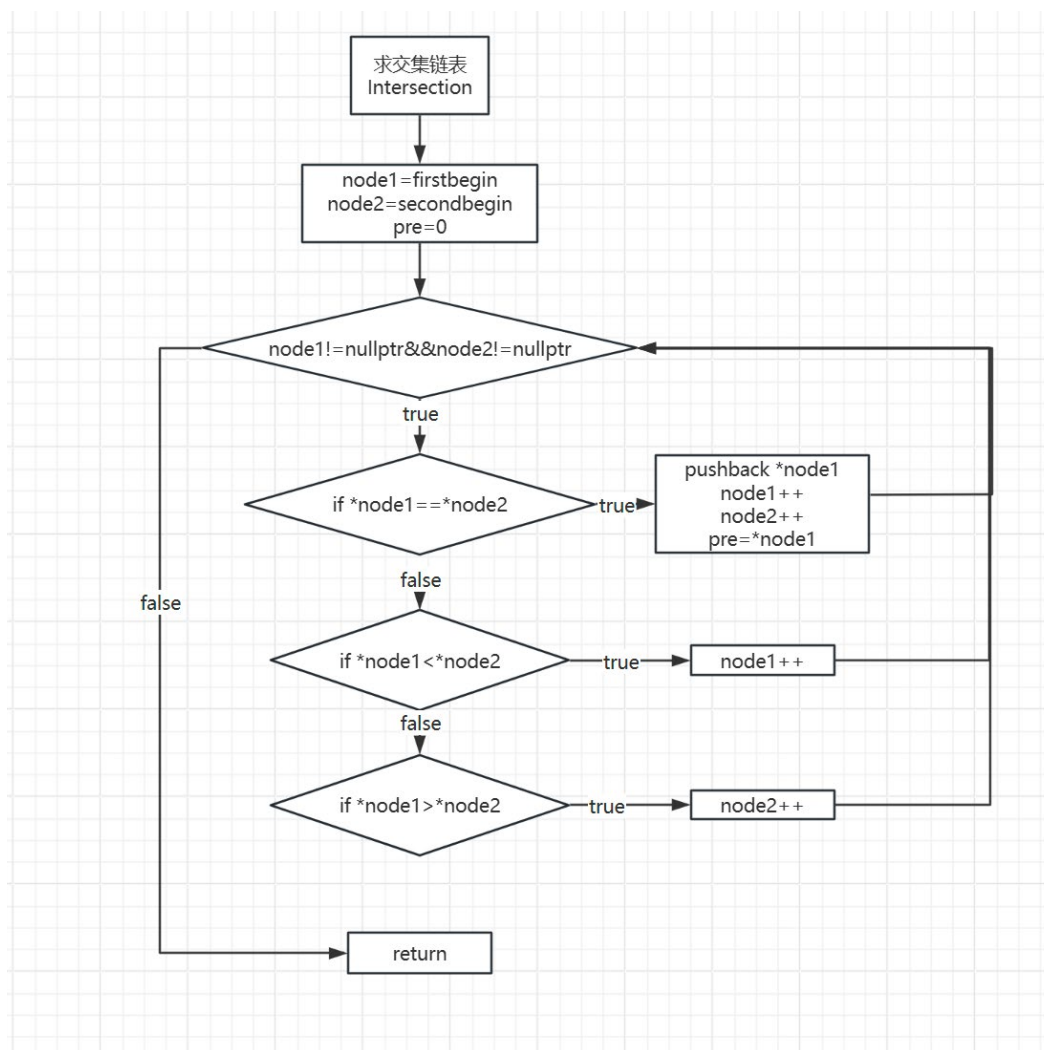
## 3.4 有序链表处理功能

### 3.4.1 实现思路

有序链表处理功能的函数名为 Intersection，求交集实现的思路为：

1. 分别获取两个链表的头结点（或头结点的下一个结点）；
2. 若链表 1 与链表 2 当前结点数据相同，且与前一个插入新链表的结点数据不同（可以使用一个变量保存数据），将其插入新链表；
3. 若结点数据不同，数据较小的链表的结点指针指向下一个结点；
4. 如此循环执行，直到某一个链表的结点指针为空指针；
5. 退出函数，返回交集链表。

### 3.4.2 流程图



### 3.4.3 核心代码

```

LinkedList<int> Intersection(LinkedList<int>& first,LinkedList<int>& second)
{
    LinkedList<int> temp;
    auto node1=first.begin();
    auto node2=second.begin();
    int pre=0;
    while(node1 != first.end()&&node2 != second.end()) {
        if(*node1==*node2) {
            temp.inputRear(*node1);
            pre==*node1;
            ++node1;
            ++node2;
        }
    }
    return temp;
}

```

```

    }
    else if(*node1<*node2)
        ++node1;
    else
        ++node2;
    }
    return temp;
}

```

### 3.4.4 示例

示例同 3.1.4

## 3.5 异常处理功能

### 3.5.1 输入非法的异常处理

在寻找交集之前，首先需要输入两个有序链表，输入链表的格式要求其为非降序正整数数列；对于每一个元素，输入时需要经历以下过程：

1. 进入一个无限 while 循环，开始输入；
2. 输入完成后，首先验证输入其是否为一个正整数或者是否为-1，不是则清除缓冲区，重新输入该元素，否则执行下一步验证；
3. 验证其是否为-1，若为-1 则退出循环及输入函数，表示当前链表输入完成，反之执行下一步验证；
4. 验证其是不小于于前一个输入的数 previous，若小于则不是则清除缓冲区，重新输入该元素；
5. 将该数放在链表尾部，previous 置为该数的值（previous 初始为 1），开始输入下一个元素。

### 3.5.2 动态内存申请失败的异常处理

在进行 Linklist 类的动态内存申请时，程序使用 new(std::nothrow) 来尝试分配内存。new(std::nothrow) 在分配内存失败时不会引发异常，而是返回一个空指针（NULL 或 nullptr），代码检查指针是否为空指针，如果为空指

针，意味着内存分配失败，对于内存分配失败，可以判断 `new(std::nothrow)` 是否返回 `nullptr`，是则使用 `exit` 函数退出程序，并限定返回值，示例：

```
template <typename T>
LinkedList<T>::LinkedList(){
    head = new(std::nothrow) Node<T>;
    if (head == nullptr) {
        std::cout << "内存分配错误！\n";
        exit(MEMORY_ALLOCATION_ERROR);
    }
}
```

## 第4章 项目测试

#### 4.1 有序链表输入功能测试

分别输入含超过上下限的整数、负数、浮点数、字符、字符串，以及不符合非降序要求的数列验证程序对输入非法的情况。

请输入第1个由若干正整数构成的非降序序列，其中-1表示输出结束  
1 2 3 999999999999999999 -1  
输入的元素包含非正整数项，请重新输入

请输入第1个由若干正整数构成的非降序序列，其中-1表示输出结束  
1 2 3 -99999999999999999999 9 -1  
输入的元素包含非正整数项，请重新输入

请输入第1个由若干正整数构成的非降序序列，其中-1表示输出结束  
1 2 3 -5 9 -1  
输入的元素包含非正整数项，请重新输入

请输入第1个由若干正整数构成的非降序序列，其中-1表示输出结束  
1 2 3 4 a 9 -1  
输入的元素包含非正整数项，请重新输入

请输入第1个由若干正整数构成的非降序序列，其中-1表示输出结束  
1 2 3 abcd 9 -1  
输入的元素包含非正整数项，请重新输入

请输入第1个由若干正整数构成的非降序序列，其中-1表示输出结束  
1 2 3 6 7 10 5 11 -1  
输入的元素中存在前项大于后项的情况，请重新输入

当输入合法时，程序继续运行

## 4.2 有序链表输出功能测试

当链表为空时，输出“NULL”，反之输出全部元素。

```

+-----+
|              有序链表序列的交集              |
| Intersection Of OrderedL inked List Sequences |
+-----+
请输入第1个由若干正整数构成的非降序序列，其中-1表示输出结束
-1
请输入第2个由若干正整数构成的非降序序列，其中-1表示输出结束
-1

第一个有序链表为：
NULL
有序链表元素总数为：0
第二个有序链表为：
NULL
有序链表元素总数为：0
两个有序链表序列的交集为：
NULL
有序链表元素总数为：0

进程已结束，退出代码为 0

```

```

请输入第1个由若干正整数构成的非降序序列，其中-1表示输出结束
1 2 3 4 5 6 -1
请输入第2个由若干正整数构成的非降序序列，其中-1表示输出结束
-1

第一个有序链表为：
1 2 3 4 5 6
有序链表元素总数为：6
第二个有序链表为：
NULL
有序链表元素总数为：0
两个有序链表序列的交集为：
NULL
有序链表元素总数为：0

```

```

+-----+
|              有序链表序列的交集              |
| Intersection Of OrderedL inked List Sequences |
+-----+
请输入第1个由若干正整数构成的非降序序列，其中-1表示输出结束
1 2 3 4 5 -1
请输入第2个由若干正整数构成的非降序序列，其中-1表示输出结束
6 7 8 9 10 -1

第一个有序链表为：
1 2 3 4 5
有序链表元素总数为：5
第二个有序链表为：
6 7 8 9 10
有序链表元素总数为：5
两个有序链表序列的交集为：
NULL
有序链表元素总数为：0

```

### 4.3 有序链表输出功能测试

当两个链表有交集时，输出交集，图同 3.1.4；

当至少有一个链表为空时或两个链表没有交集时，交集链表为空，图同 4.2。

## 第 5 章 相关说明

### 5.1 编程语言

本项目全部 .cpp 文件以及 .h 文件均使用 C++ 编译完成。

### 5.2 Windows 环境

Windows 系统: Windows 11 x64

Windows 集成开发环境: CLion 2024

工具集: MinGW 11.0 w64

### 5.3 Linux 环境

基于 Linux 内核的操作系统发行版: Ubuntu 24.04.1

Linux 命令编译过程为:

1. 定位包含项目所在文件夹, 包括 .pp 与 .h 文件; 具体命令为: `cd /home/bruce/programe/Intersection_of_ordered_linked_list_sequences`

2. 编译项目, 生成可执行文件; 具体命令为: `g++ -static -o Intersection_of_ordered_linked_list_sequences_linux Intersection_of_ordered_linked_list_sequences.cpp my_singly_link_list.h;`

其中指令含义分别为:

`g++`: 调用 GNU 的 C++ 编译器

`-static`: 使用静态链接而非动态链接, 将所有依赖库直接嵌入到可执行文件, 文件存储空间变大, 但可以单独运行

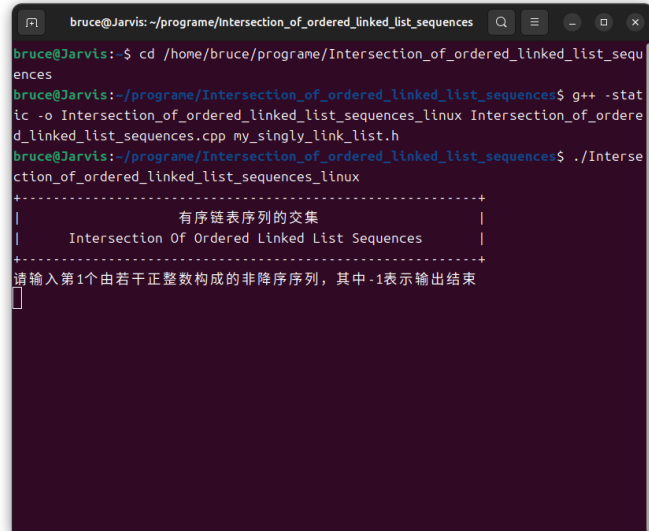
`-o Intersection_of_ordered_linked_list_sequences_linux`: `-o` 表示输出文件选项, `Intersection_of_ordered_linked_list_sequences_linux` 为可执行文件名

`Intersection_of_ordered_linked_list_sequences.cpp`  
`my_singly_link_list.h`: 编译所需要的文件。



3. 运行可执行文件；具体命令为 ./

Intersection\_of\_ordered\_linked\_list\_sequences\_linux



```
bruce@Jarvis: ~/programe/Intersection_of_ordered_linked_list_sequences
bruce@Jarvis:~/programe/Intersection_of_ordered_linked_list_sequences$ cd /home/bruce/programe/Intersection_of_ordered_linked_list_sequences
bruce@Jarvis:~/programe/Intersection_of_ordered_linked_list_sequences$ g++ -static -o Intersection_of_ordered_linked_list_sequences_linux Intersection_of_ordered_linked_list_sequences.cpp my_singly_linked_list.h
bruce@Jarvis:~/programe/Intersection_of_ordered_linked_list_sequences$ ./Intersection_of_ordered_linked_list_sequences_linux
+-----+
|              有序链表序列的交集              |
|      Intersection Of Ordered Linked List Sequences      |
+-----+
请输入第1个由若干正整数构成的非降序序列，其中-1表示输出结束
□
```