数据结构课程设计

项目说明文档

两个有序链表序列的交集

软件工程

张靖凯

2151396

目录

项目要求

项目内容

已知两个非降序链表序列 S1 和 S2，设计函数构造出 S1 和 S2 的交集新链表 S3。

项目功能要求：(要求采用链表)

1. 输入说明：输入分 2 行，分别在每行给出由若干个正整数构成的非降序序列，用-1 表示序列的结尾（-1 不属于这个序列）。数字用空格间隔。
2. 输出说明：在一行中输出两个输入序列的交集序列，数字间用空格分开，结尾不能有多余空格；若新链表为空，输出 NULL。
3. 测试用例：

序号	输入	输出	说明
1	1 2 5 -1 2 4 5 8 10 -1	2 5	一般情况
2	1 3 5 -1 2 4 6 8 10 -1	NULL	交集为空的情况
3	1 2 3 4 5 -1 1 2 3 4 5 -1	1 2 3 4 5	完全相交的情况
4	3 5 7 -1 2 3 4 5 6 7 8 -1	3 5 7	其中一个序列完全属于交集的情况
5	-1 10 100 1000 -1	NULL	其中一个序列为空的情况

项目设计

数据结构设计

使用配有迭代器的双向循环链表，时间复杂度低，效率高。

合并链表的算法为依次比较两个链表中各自节点数据的大小，当是交集元素时按照非降序列放入第三个链表中，算法时间复杂度为 O(n+m)。

插入	删除	访问
push_front O(1)	pop_front O(1)	O(n)
push_back O(1)	pop_back O(1)	
insert O(1)	erase O(1)	

类设计说明

命名空间 MercedesKK 中共三个类，另有本题中的 Student 类。

MercedesKK	
List	仿STL中的List
ListIterator	用迭代器封装指针，重载++ – ()等operator 同时迭代器种类设置为bidirectional_iterator_tag
Node	结点类

Node 类

Node 的对象为 list 中的每个节点，其中有成员 value_type 类型的数据和 NodePtr 类型的前后指针，默认构造函数初始化均指向自身。

Public Types

using	value_type = T
using	NodePtr = Node< T > *
using	NodeRef = Node< T > &

Public Attributes

value_type	_data 元素 More...
NodePtr	_next 后指针 More...
NodePtr	_prev 前指针 More...

ListIterator 类

ListIterator 仿照 STL，是 List 配有的迭代器，其中为了满足 traits 特性，将类型别名规范；同时设置 iterator category 为 bidirectional_iterator_tag 类型；封装指针过程中重载了自增自减等运算符，同时还搭配了 const 类型的重载。

Public Types

using	**LinkNode** = **Node**< T >
using	**self** = **ListIterator**< T, Ref, Ptr >
using	**iterator_category** = std::bidirectional_iterator_tag
using	**value_type** = T
using	**pointer** = Ptr
using	**reference** = Ref
using	**size_type** = size_t
using	**difference_type** = ptrdiff_t
using	**const_reference** = const Ref
using	**const_pointer** = const Ptr

Public Member Functions

	ListIterator (**LinkNode** *pnode=nullptr) 指针转迭代器构造函数 More...
	ListIterator (const **self** &It) 拷贝构造函数 More...
reference	**operator*** () 重载* More...
pointer	**operator->** () 重载-> More...
const_reference	**operator*** () const 重载* const类型 More...
const_pointer	**operator->** () const 重载-> const类型 More...
bool	**operator!=** (const **self** &x) const 重载!= More...
bool	**operator==** (const **self** &x) const 重载== More...
self &	**operator++** ()
self	**operator++** (int)
self &	**operator--** ()
self	**operator--** (int)

List 类

本类仿照 STL 中的 list，为双向循环链表。同样使用了标准 STL 中的类型别名。实现了 const 和非 const 迭代器。构造函数有迭代器区间构造函数，构造指定 n 个元素函数,拷贝构造函数。重载了多个 insert 函数，包括插入一个数据，多个数据，通过迭代器插入数据等等，erase 函数同理。也提供了同 STL 中的 push_back 等函数，在内部实现是统一使用 insert 进行处理。

Public Types

using **value_type** = T

using **difference_type** = ptrdiff_t

using **size_type** = size_t

using **pointer** = **value_type** *

using **reference** = **value_type** &

using **LinkNode** = **Node**< T >

using **iterator** = **ListIterator**< T, T &, T * >

using **const_iterator** = **ListIterator**< T, const T &, const T * >

Public Member Functions

iterator **begin** () noexcept
头迭代器 More...

iterator **end** () noexcept
尾迭代器 More...

const_iterator **cbegin** () const noexcept
const 头迭代器 More...

const_iterator **cend** () const noexcept
const 尾迭代器 More...

List< T > & **operator=** (**List**< T > &lt)
赋值运算符重载 More...

void **push_back** (const T &val)
尾插 More...

void **push_front** (const T &val)
头插 More...

void **pop_back** ()
尾删 More...

void **pop_front** ()
头删 More...

iterator **insert** (**iterator** pos, const T &val=T())
任意位置插入一个数据 More...

void **insert** (**iterator** pos, int n, const T &val)
任意位置插入多个数据 More...

template<class InputIterator >
void **insert** (**iterator** pos, InputIterator first, InputIterator last)
任意位置插入一个迭代器区间的数据 More...

iterator **erase** (**iterator** pos)
任意位置删除一个数据 More...

iterator **erase** (**iterator** first, **iterator** last)
删除一个迭代器区间的数据[first, last) More...

iterator **find** (**iterator** first, **iterator** last, T element)
查找元素 More...

int **find** (const T &element)
查找元素 More...

bool **empty** ()
判空 More...

size_t **size** ()
计算个数 More...

void **clear** ()
清除 More...

void **swap** (**List**< T > &lt)
交换 More...

List () noexcept
无参构造函数 More...

template<class InputIterator >
List (InputIterator first, InputIterator last) noexcept
迭代器区间构造函数 More...

List (int n, const T &val=T()) noexcept
构造指定n个元素函数 More...

List (const **List**< T > &lt) noexcept
拷贝构造函数 More...

~List ()
析构函数 More...

Private Attributes

LinkNode * **_head**
头指针 More...

解题思路

两个链表的指针指向的节点比较大小，依次将指针后移，将数据存入新建链表中，时间复杂度为 O(n+m)

关键代码

```
auto it1 = list1.begin();
auto it2 = list2.begin();
while (it1 != list1.end() && it2 != list2.end())
{
    if (*it1 < *it2)
        ++it1;
    else if (*it1 > *it2)
        ++it2;
    else
    {
        list3.push_back(*it1);
        ++it1;
        ++it2;
    }
}
```

设计亮点

代码注释规范

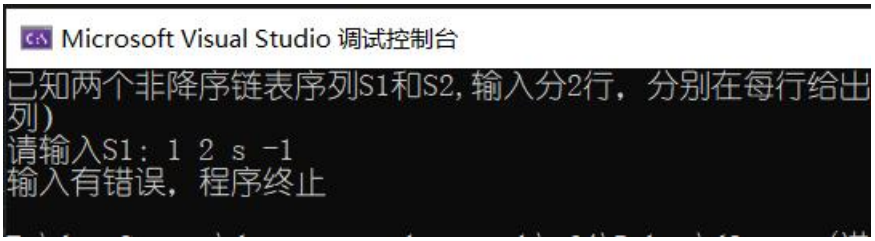
采用了 doxygen 注释规范，对类、函数等有简要说明，命名采用驼峰命名法，类内的各个声明规范，方便本人回顾之前写过的代码和 code reviewer 查看，使 API 规范，帮助这个开发流程高效、规范地进行。

```cpp
/**
* @class ListIterator
* @brief 用迭代器封装指针，重载++ -- ()等operator
* 同时迭代器种类设置为bidirectional_iterator_tag
*/
template<class T, class Ref, class Ptr>
class ListIterator
{
public:
    using LinkNode          = Node<T>;
    using self              = ListIterator<T, Ref, Ptr>;

    using iterator_category = std::bidirectional_iterator_tag;
    using value_type        = T;
    using pointer           = Ptr;
    using reference         = Ref;
    using size_type         = size_t;
    using difference_type   = ptrdiff_t;
```

```cpp
public:
    iterator begin()       noexcept { return _head->_next; }          ///< 头迭代器
    iterator end()         noexcept { return _head; }                 ///< 尾迭代器
    const_iterator cbegin()const noexcept { return _head->_next; }    ///< const 头迭代器
    const_iterator cend()  const noexcept { return _head; }           ///< const 尾迭代器
```

输入错误处理



项目测试

交集为空的情况

一般情况

完全交集的情况

```
请输入S1和S2: 1 2 3 4 -1
1 2 3 4 -1
合并后的链表: 1 2 3 4
Enter to Exit
```

一个序列为空的情况

```
请输入S1和S2: -1 2 3 4 -1
合并后的链表: NULL
Enter to Exit
```

Linux 测试（Ubuntu）

kk@LAPTOP-UJDPHKT8: ~/da

```cpp
#include <iostream>
#include "List.hpp"
#include <cassert>

using namespace MercedesKK;
using std::cin;
using std::cout;
using std::endl;

/// @brief 两个废降序链表合成一条链表
/// @param>->---list1 第一个链表
/// @param>->---list2 第二个链表
/// @return 返回合并后的链表
List<int>& mergeList(List<int>& list1, List<int>& list2>,List<int>& list3)
{
    auto it1 = list1.begin();
    auto it2 = list2.begin();

    while (it1 != list1.end() && it2 != list2.end())
    {
        if (*it1 < *it2)
            ++it1;
        else if (*it1 > *it2)
            ++it2;
        else
        {
            list3.push_back(*it1);
```

h2.cpp
"h2.cpp" 99L, 2100C

```
kk@LAPTOP-UJDPHKT8:~/dataStruct/homework5$ vim h5.cpp
kk@LAPTOP-UJDPHKT8:~/dataStruct/homework5$ cd ..
kk@LAPTOP-UJDPHKT8:~/dataStruct$ ls
homework1  homework2  homework3  homework4  homework5  homework6  homework7  test
kk@LAPTOP-UJDPHKT8:~/dataStruct$ cd homework2
kk@LAPTOP-UJDPHKT8:~/dataStruct/homework2$ ls
List.hpp  h2.cpp  run
kk@LAPTOP-UJDPHKT8:~/dataStruct/homework2$ ./run
已知两个非降序链表序列S1和S2,输入分2行，分别在每行给出由若干个正整数构成的非降序序列，
列)
请输入S1和S2: 1 2 3 4 -1
2 3 5 6 -1
合并后的链表: 2 3
Enter to Exit
```