

**数据结构课程实验报告**

**（4）**

**姓名：王秋实**

**学号：2135070824**

**专业：计科**

**完成日期：2023/4/12**

**目录**

[目录 - 1 -](#_Toc434170429)

[1 设计要求 - 2 -](#_Toc434170430)

[2 程序功能框图 - 2 -](#_Toc434170431)

[3 数据结构说明 - 2 -](#_Toc434170432)

[4 重要算法核心代码 - 2 -](#_Toc434170433)

[5 测试运行界面 - 2 -](#_Toc434170434)

[6 完整源程序 - 3 -](#_Toc434170435)

**数据结构实验四（综合练习）**

**1、链表**

【问题描述】

给定两个已升序排序的链表L1和L2，只使用基本的链表操作编写计算L1∩L2的过程。结果链表也要求是升序排序的。

**2、链表/堆栈**

【问题描述】

用两种非递归方法以O(N)的时间复杂度，实现反转一个单链表。

* 做法1：用堆栈来保存链表的遍历，用pop倒序输出，每倒序输出一个值就构造新链表的一个结点。
* 做法2：在原链表上对结点指针重新赋值操作来进行链表的反转。

**3、队列**

【问题描述】

假设以数组Q[m]存放循环队列中的元素，同时以rear和length 分别指示环形队列中的队尾位置和队列中所含元素的个数：

1）求队列中第一个元素的实际位置。

2）给出该循环队列的队空条件和队满条件，并写出相应的插入(enqueue)和删除(delqueue)元素的操作。

**4、二叉树**

【问题描述】

判断一个二叉树是不是二叉搜索树。

二叉搜索树必须满足：

1)或者是空树

2)如果不是空树，则对于任意一个节点

* 左子树（如果有）中所有节点都小于该节点
* 右子树（如果有）中所有节点都大于该节点
* 左子树（如果有）也是二叉搜索树
* 右子树（如果有）也是二叉搜索树

**2、程序功能框图**

1、链表：

输入格式：第一行两个整数n m，表示两个链表分别有n项和m项，接下来的两行分别有n和m个数，第k行第i个数表示第k-1个链表的第i项的值

输出格式：共一行，输出两个链表相同的元素，用空格分开

2、链表/堆栈：

输入格式：第一行一个整数n，表示链表共有n项，第二行n个整数，表示链表的n项。

输出格式：共一行，将链表翻转后输出链表，用空格分开

3、队列：

输入格式：第一行一个整数n，表示链表共有n项，第二行n个整数，表示链表的n项。

输出格式：输出队头位置，是否队空，是否队满

4、二叉树

输入格式：第一行一个整数n，接下来的n行中，第i行有三个整数，分别表示序号为i的结点的结点值，左子树和右子树的结点序号

输出格式：一行，是二叉搜索树输出yes，不是输出no

**3、数据结构说明**

1、链表

开了三个链表，分别表示两个初始链表和结果链表，两个指针指向链表的下一个元素，遍历整个链表即可找出相同的元素

2、链表/堆栈：

使用一个双向循环链表，表示初始的元素序列，然后将链表的元素从尾到头依次弹出到另一个链表中，从而实现链表的翻转

3、队列

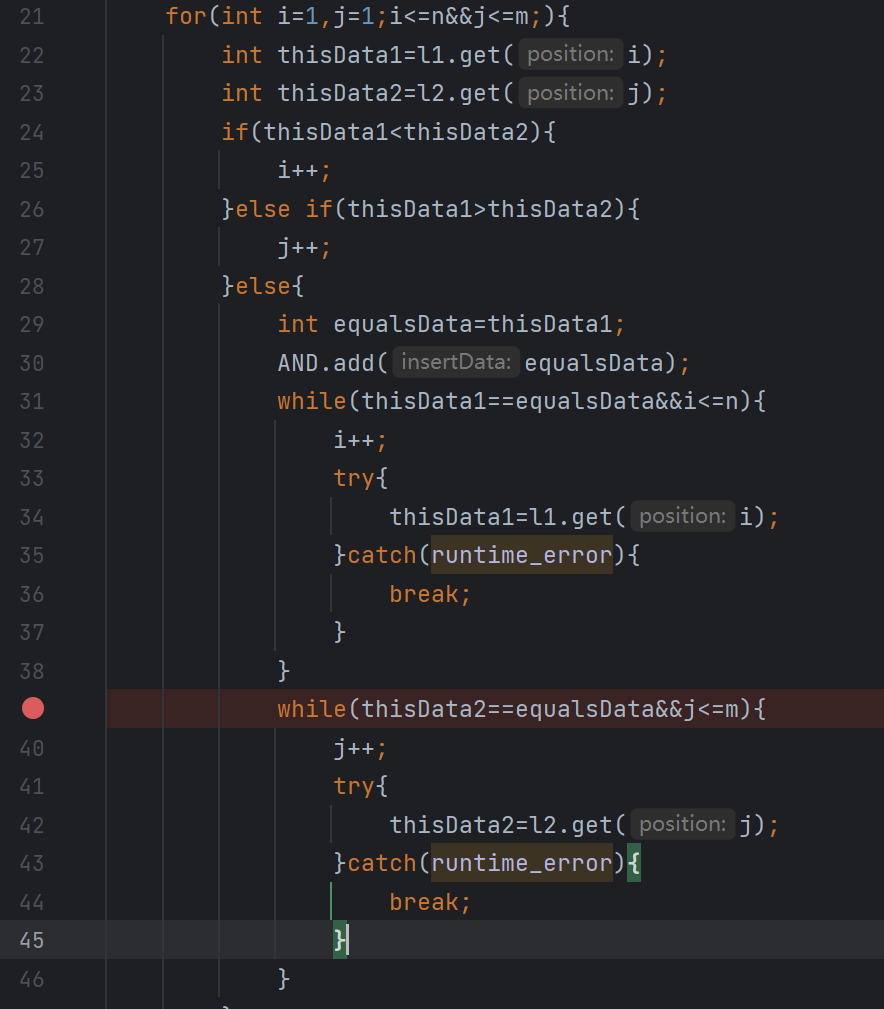
开一个队列，用一个rear指针指向队列下一个位置，用length表示队列的长度，队列的front指针用(rear-length+maxLength)%maxLength表示

4、二叉树

开一个二叉树，中序遍历二叉树并输出元素值，然后判断中序序列是否为严格单调递增序列

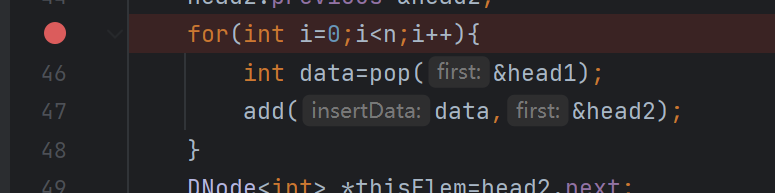
**4、重要算法核心代码**

1、链表



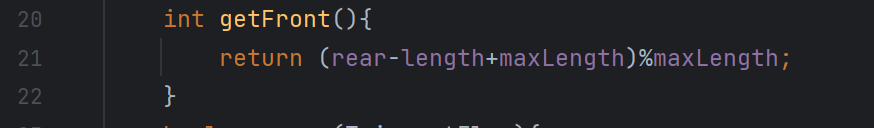
从头到尾遍历整个链表，用两个指针指向下一个要访问的元素thisData1和thisData2，如果1比2大，则2的指针右移，如果2比1大，则1的指针右移，如果1和2相等，则输出该值并将1和2的指针移向下一个比这个数大的值的位置

2、链表/堆栈

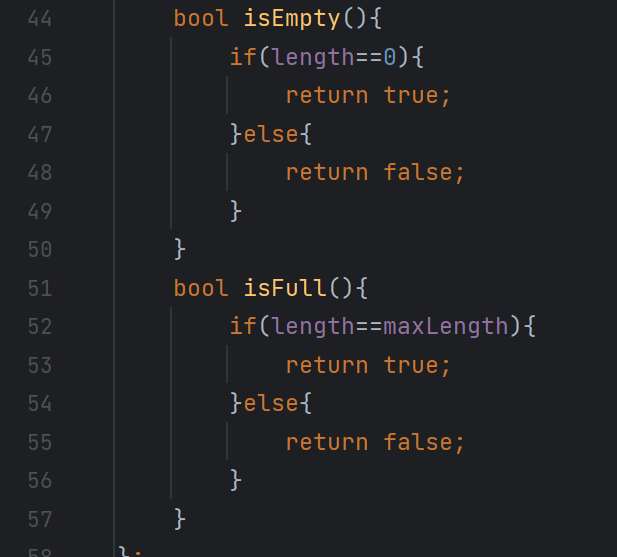


把链表模拟成栈，依次弹出链表的元素并压入另一个链表中

3、队列



计算front指针的方法



判队空队满的方法

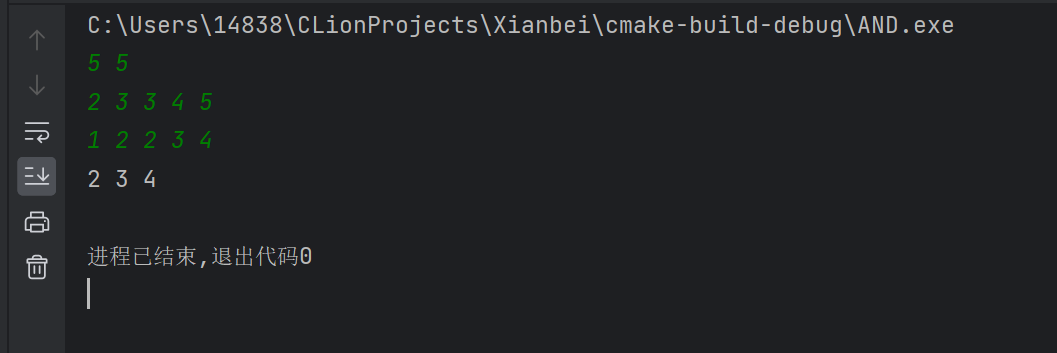
4、二叉树



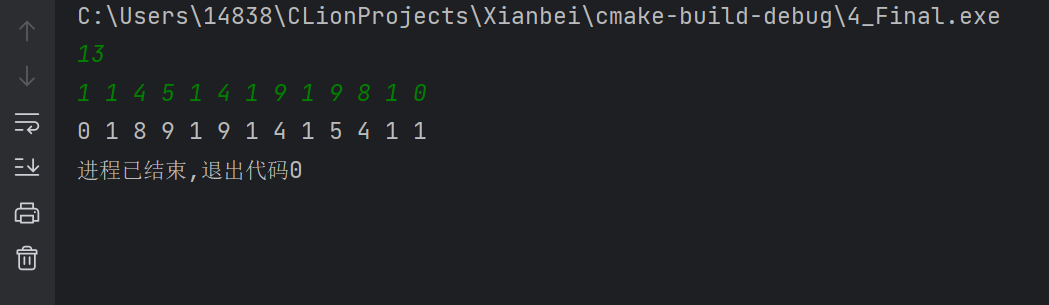
对二叉树中序遍历，依次输出中序遍历结果

**4、测试运行界面**

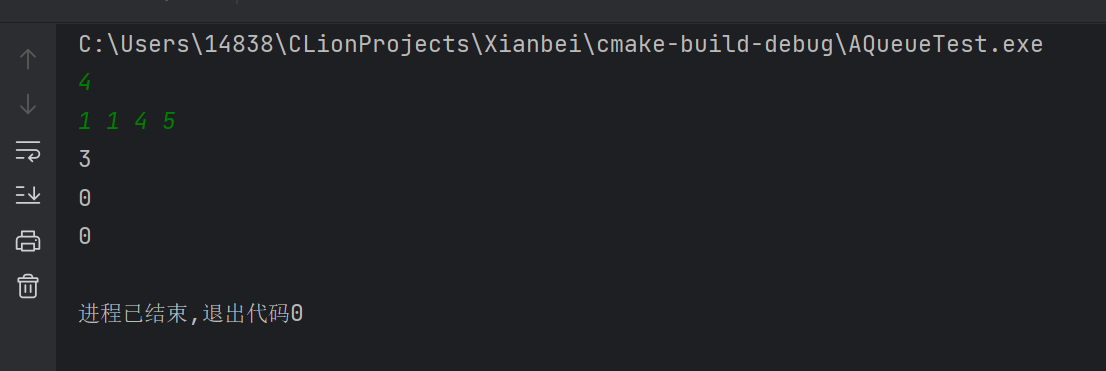
1、链表：



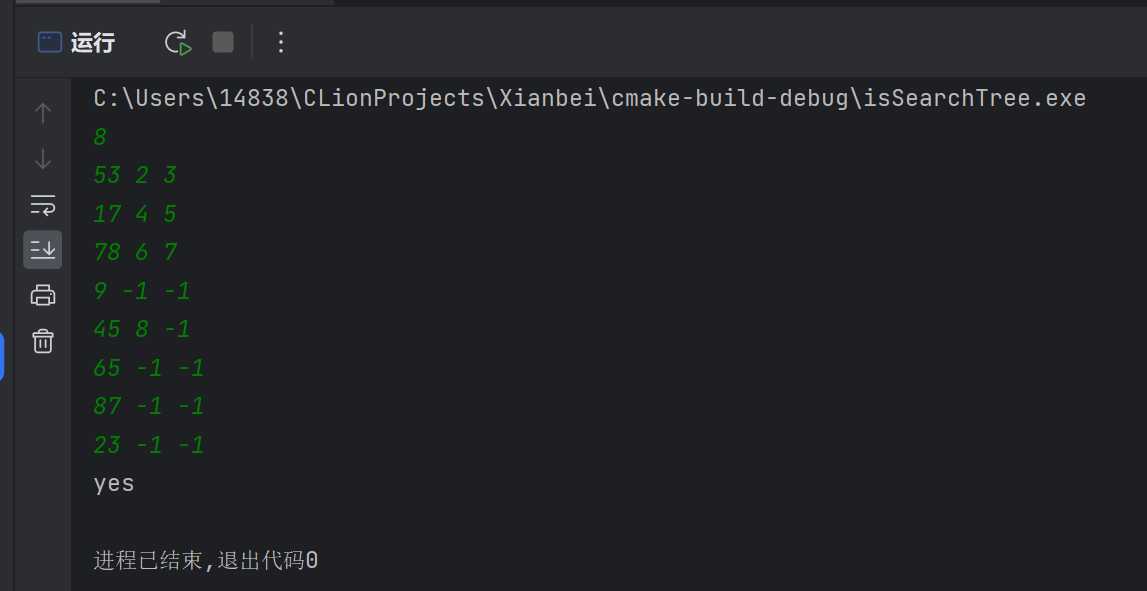
2、链表/队列：

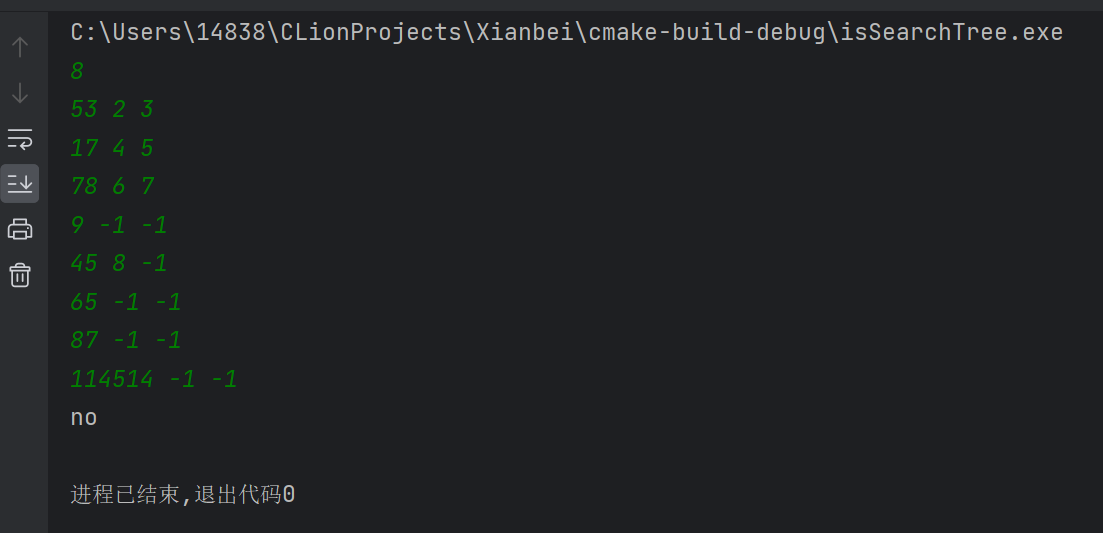


3、队列



4、二叉树





**5、源代码**

1、链表

#pragma once

#ifndef XIANBEI\_DLINKEDLIST\_H

#define XIANBEI\_DLINKEDLIST\_H

#include "iostream"

#include "exception"

template<typename T>

struct DNode{

T data;

DNode<T> \*previous=NULL;

DNode<T> \*next=NULL;

};

template<typename T>

class DLinkedList {

public:

int length;

DLinkedList(){ //构造函数

length=0;

first=new DNode<T>;

}

~DLinkedList(){ //析构函数

clear();

if(first!=NULL){

delete first;

first=NULL;

}

}

void print(){ //打印链表数据

DNode<T> \*thisElem;

thisElem=first->next;

while(thisElem!=NULL){

std::cout<<thisElem->data<<" ";

thisElem=thisElem->next;

}

std::cout<<std::endl;

}

bool add(int position,T &insertData){ //后插操作

if(position>length||position<=0){

return false;

}

DNode<T> \*thisElem=first;

for(int i=0;i<position;i++){

thisElem=thisElem->next;

}

DNode<T> \*insertElem=new DNode<T>;

insertElem->data=insertData;

insertElem->previous=thisElem;

insertElem->next=thisElem->next;

thisElem->next=insertElem;

if(position==length){

thisElem->next->previous=insertElem;

}

length++;

return true;

}

bool add(T insertData){ //尾插操作

DNode<T> \*thisElem=first;

for(int i=0;i<length;i++){

thisElem=thisElem->next;

}

DNode<T> \*insertElem=new DNode<T>;

insertElem->data=insertData;

thisElem->next=insertElem;

insertElem->previous=thisElem;

length++;

return true;

}

bool remove(int position){ //给定位置删除

if(position>length||length==0||position<=0){

return false;

}

DNode<T> \*delElem=first;

for(int i=0;i<position;i++){

delElem=delElem->next;

}

if(position==length){

DNode<T> \*previousElem=delElem->previous;

delete delElem;

previousElem->next=NULL;

}else{

DNode<T> \*nextElem=delElem->next;

DNode<T> \*previousElem=delElem->previous;

delete delElem;

previousElem->next=nextElem;

nextElem->previous=previousElem;

}

length--;

return true;

}

bool replace(int position,T &insertData){ //给定位置修改值

if(position>length||length==0||position<=0){

return false;

}

DNode<T> \*replaceElem=first;

for(int i=0;i<position;i++){

replaceElem=replaceElem->next;

}

replaceElem->data=insertData;

return true;

}

bool isEmpty(){ //判空

if(length==0){

return true;

}else{

return false;

}

}

bool clear(){ //清空链表

if(length==0){

return false;

}

DNode<T> \*delElem=first;

while(delElem->next!=NULL){

delElem=delElem->next;

}

while(delElem->previous!=NULL){

delElem=delElem->previous;

delete delElem->next;

}

delElem->next=NULL;

length=0;

return true;

}

T get(int position){

DNode<T> \*getElem=first;

if(position>length||length==0||position<=0){

throw std::runtime\_error("IndexOutOfBounds");

}

for(int i=0;i<position;i++){

getElem=getElem->next;

}

return getElem->data;

}

private:

DNode<T> \*first;

};

#endif //XIANBEI\_DLINKEDLIST\_H

#include "iostream"

#include "DLinkedList.h"

using namespace std;

int main(){

int n,m;

DLinkedList<int> l1;

DLinkedList<int> l2;

cin>>n>>m;

for(int i=0;i<n;i++){

int thisData;

cin>>thisData;

l1.add(thisData);

}

for(int i=0;i<m;i++){

int thisData;

cin>>thisData;

l2.add(thisData);

}

DLinkedList<int> AND;

for(int i=1,j=1;i<=n&&j<=m;){

int thisData1=l1.get(i);

int thisData2=l2.get(j);

if(thisData1<thisData2){

i++;

}else if(thisData1>thisData2){

j++;

}else{

int equalsData=thisData1;

AND.add(equalsData);

while(thisData1==equalsData&&i<=n){

i++;

try{

thisData1=l1.get(i);

}catch(runtime\_error){

break;

}

}

while(thisData2==equalsData&&j<=m){

j++;

try{

thisData2=l2.get(j);

}catch(runtime\_error){

break;

}

}

}

}

AND.print();

}

2#、链表/堆栈

#include "iostream"

using namespace std;

template<typename T>

struct DNode {

T data;

DNode<T> \*previous = NULL;

DNode<T> \*next = NULL;

};

template<typename T>

bool add(T insertData,DNode<T> \*first){ //尾插操作

DNode<T> \*thisElem=first->previous;

DNode<T> \*insertElem=new DNode<T>;

insertElem->data=insertData;

insertElem->previous=thisElem;

insertElem->next=first;

thisElem->next=insertElem;

first->previous=insertElem;

return true;

}

template<typename T>

T pop(DNode<T> \*first){

T getData=first->previous->data;

DNode<T> \*lastElem=first->previous->previous;

delete lastElem->next;

lastElem->next=first;

first->previous=lastElem;

return getData;

}

int main(){

int n;

cin>>n;

DNode<int> head1;

head1.next=&head1;

head1.previous=&head1;

for(int i=0;i<n;i++){

int thisData;

cin>>thisData;

add(thisData,&head1);

}

DNode<int> head2;

head2.next=&head2;

head2.previous=&head2;

for(int i=0;i<n;i++){

int data=pop(&head1);

add(data,&head2);

}

DNode<int> \*thisElem=head2.next;

for(int i=0;i<n;i++){

cout<<thisElem->data<<" ";

thisElem=thisElem->next;

}

}

3、队列

#ifndef XIANBEI\_AQUEUE\_H

#define XIANBEI\_AQUEUE\_H

#include "iostream"

#include "exception"

template <typename T>

class AQueue{ //都什么年代了，还用传统顺序队列

private:

const static int maxLength=5;

T elem[maxLength];

int rear;

public:

int length;

AQueue(){

rear=0;

length=0;

}

int getFront(){

return (rear-length+maxLength)%maxLength;

}

bool enqueue(T insertElem){

if(isFull()){

return false;

}

elem[rear]=insertElem;

rear=(rear+1)%maxLength;

length++;

return true;

}

T dequeue(){

T popElem=peek();

length--;

int front=getFront();

return popElem;

}

T peek(){

if(isEmpty()){

throw std::runtime\_error("IndexOutOfBounds");

}

return elem[getFront()];

}

bool isEmpty(){

if(length==0){

return true;

}else{

return false;

}

}

bool isFull(){

if(length==maxLength){

return true;

}else{

return false;

}

}

};

#endif //XIANBEI\_AQUEUE\_H

#include "AQueue.h"

#include "iostream"

using namespace std;

int main(){

int n;

cin>>n;

AQueue<int> queue;

for(int i=0;i<n;i++){

int thisData;

cin>>thisData;

queue.enqueue(thisData);

}

queue.dequeue();

queue.dequeue();

queue.dequeue();

//找到front指针：

cout<<queue.getFront()<<endl;

//队空条件：length==0

cout<<queue.isEmpty()<<endl;

//队满条件：length==maxLength

cout<<queue.isFull()<<endl;

}

4、二叉树

#include "iostream"

#define MAXN 114514

#define MINN -114514

using namespace std;

class Node{

public:

int id;

int data;

int l=-1;

int r=-1;

Node(int id,int data,int l,int r){

this->data=data;

this->id=id;

this->l=l;

this->r=r;

}

Node(){}

};

int mid[MAXN];

bool isFill[MAXN];

int cnt=0;

Node sTree[MAXN];

void dfs(int thisPosition){

Node thisNode=sTree[thisPosition];

if(thisNode.l!=-1){

dfs(thisNode.l);

}

mid[cnt]=thisNode.data;

isFill[cnt]=true;

cnt++;

if(thisNode.r!=-1){

dfs(thisNode.r);

}

}

int main(){

for(int i=0;i<MAXN;i++){

isFill[i]=false;

}

int n;

cin>>n;

for(int i=1;i<=n;i++){

int data,l,r;

cin>>data>>l>>r;

sTree[i]=Node(i,data,l,r);

}

dfs(1);

bool isSearchTree=true;

for(int i=1;i<MAXN;i++){

if(isFill[i]){

if(mid[i]<=mid[i-1]){

isSearchTree=false;

break;

}

}

}

if(isSearchTree){

cout<<"yes"<<endl;

}else{

cout<<"no"<<endl;

}

}