|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 姓名 |  | 学号 |  | 实验成绩 |  |
| 专业班级 |  | 实验时间 | 2025-04-07 | 实验地点 | 第五机房 |

实验一 顺序表实验

1实验目的

1. 熟练掌握线性表的顺序存储结构。
2. 熟练掌握顺序表的有关算法设计和实现。
3. 根据具体问题的需要，设计出合理的表示数据元素的顺序结构，并设计相关算法。

2 实验要求

1. 顺序表结构和运算定义，算法的实现以库文件方式实现，不得在测试主程序中直接实现；比如存储、算法实现放入文件：seqList.h
2. 程序有适当的注释，程序的书写要采用缩进格式。
3. 实验程序有较好可读性，各运算和变量的命名直观易懂，符合软件工程要求；
4. 程序要具有一定的健壮性，即当输入数据非法时，程序也能适当地做出反应，如插入删除时指定的位置不对等等。
5. 程序要做到界面友好，在程序运行时用户可以根据相应的提示信息进行操作。
6. 程序运行、测试正确；
7. 根据实验报告模板详细书写实验报告。

3 实验任务

**编写算法实现下列问题的求解。**

1. **在一个递增有序的顺序表L中插入一个值为x的元素，并保持其递增有序特性。实验测试数据基本要求：**

**顺序表元素为 （10,20,30,40,50,60,70,80,90,100）,**

**x分别为25，85，110和8**

* 算法设计与实现描述

（书上给出的基本运算外，其它问题必须先给出算法思想或步骤，再给出算法描述）

这个算法不用详细描述，看InsertUpdate函数显然，通过遍历，比较大小

// 插入

Status Insert(List \*L,ElemType E,int place) {

L->count++;

if(L->count > MAXSIZE)

return ERROR;

int ptr = L->count;

while(ptr != place) {

L->data[ptr - 1] = L->data[ptr - 2];

ptr--;

}

L->data[ptr - 1] = E;

return OK;

}

// 列表递增 设计算法插入元素，仍保持有序，且插入后无重复元素，有则不插入

void InsertUpdate(List \*L,ElemType E) {

int ptr = 0;

while(ptr != L->count) {

if(L->data[ptr] < E) {

ptr++;

} else if(L->data[ptr] > E) {

break;

} else {

return;

}

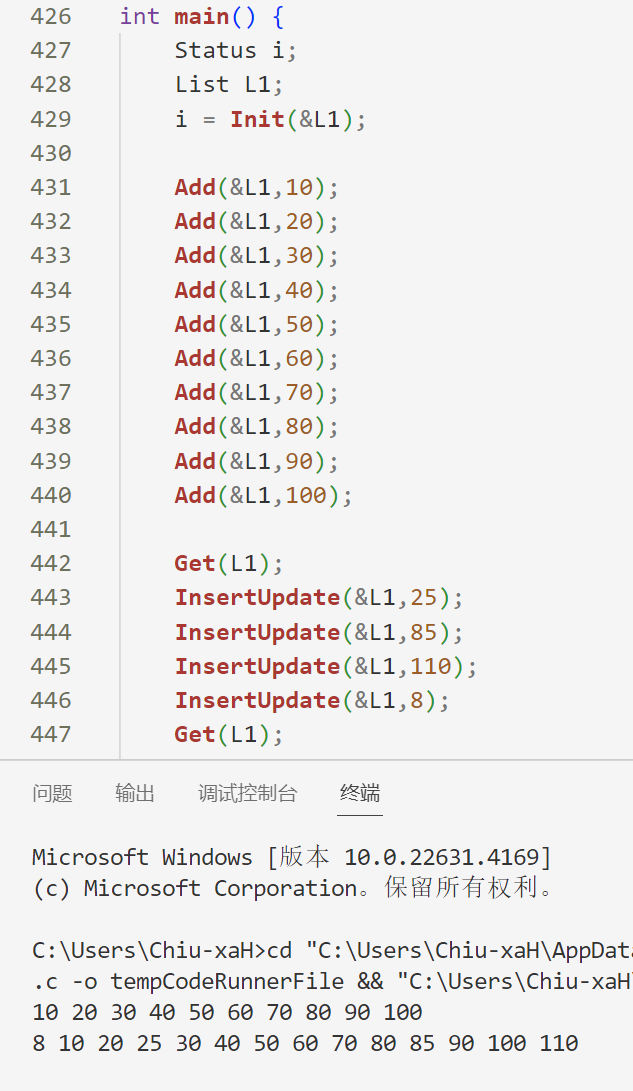
}

Insert(L,E,ptr+1);

}

* 运行结果截图及说明

测试代码及其运行结果如下



1. **将顺序表Ｌ中的奇数项和偶数项结点分解开（元素值为奇数、偶数），分别放入新的顺序表中，然后原表和新表元素同时输出到屏幕上，以便对照求解结果。实验测试数据基本要求：**

**第一组数据：顺序表元素为 （1,2,3,4,5,6,7,8,9,10,20,30,40,50,60）**

**第二组数据：顺序表元素为 （10,20,30,40,50,60,70,80,90,100）**

* 算法设计与实现描述

（书上给出的基本运算外，其它问题必须先给出算法思想或步骤，再给出算法描述）

这个也不用详细描述，看函数，就是取余判断奇偶，然后Add

Status Add(List \*L,ElemType E) {

if(L->count == MAXSIZE) {

return ERROR;

}

L->data[L->count] = E;

L->count++;

return OK;

}

//设计算法分解链表为奇数表、偶数表

void Resolve(List L,List \*JiL,List \*OuL) {

Init(JiL);

Init(OuL);

int ptr = 0;

while(ptr != L.count) {

int item = L.data[ptr];

if(item % 2 == 0) {

Add(OuL,item);

} else {

Add(JiL,item);

}

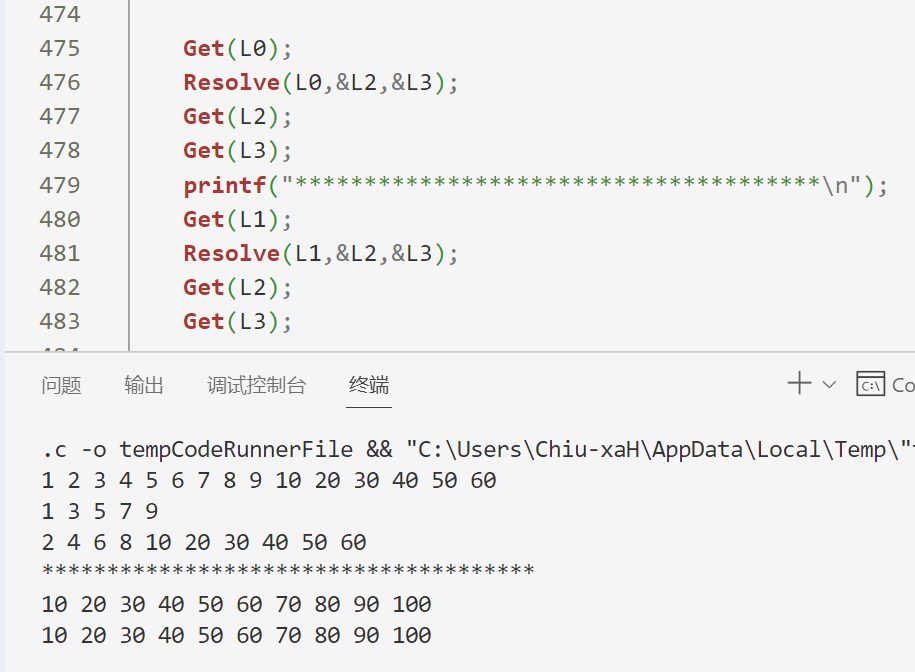
ptr++;

}

}

* 运行结果截图及说明

测试代码与运行结果如下，L0为第一组源数据，L1为第二组原数据，L2为奇数表（最后数据未输出是因为第二组源数据没有奇数），L3为偶数表



1. **求两个递增有序顺序表L1和L2中的公共元素，放入新的顺序表L3中。**

**实验测试数据基本要求：**

**第一组**

**第一个顺序表元素为 （1，3，6，10，15，16，17，18，19，20）**

**第二个顺序表元素为 （1，2，3，4，5，6，7，8，9，10，18，20，30）**

**第二组**

**第一个顺序表元素为 （1，3，6，10，15，16，17，18，19，20）**

**第二个顺序表元素为 （2，4，5，7，8，9，12，22）**

**第三组**

**第一个顺序表元素为 （）**

**第二个顺序表元素为 （1，2，3，4，5，6，7，8，9，10）**

* 算法设计与实现描述

（书上给出的基本运算外，其它问题必须先给出算法思想或步骤，再给出算法描述）

分析题目，就是两个递增有序顺序表的交集，直接拿写好的函数

List JointSet(List list1,List list2) {

int ptr1 = 0,ptr2 = 0;

List new;

Init(&new);

while (ptr1 != list1.count && ptr2 != list2.count) {

int data1 = list1.data[ptr1];

int data2 = list2.data[ptr2];

if(data1 > data2) {

ptr2++;

} else if(data1 < data2) {

ptr1++;

} else {

new.data[new.count] = data1;

new.count++;

ptr1++;

ptr2++;

}

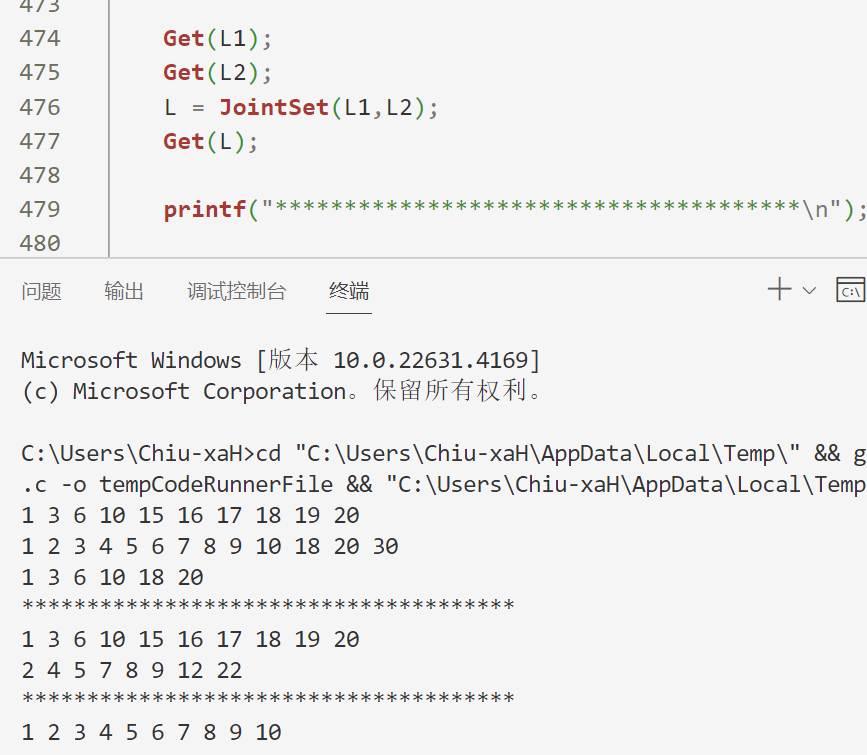
}

return new;

}

* 运行结果截图及说明

测试代码与运行结果如下，L1为数据表1，L2数据表2，L为交集表（第二组和第三组数据未输出是因为无交集）



1. **删除递增有序顺序表中的重复元素，并统计移动元素次数，要求时间性能最好。实验测试数据基本要求：**

**第一组数据：顺序表元素为 （1,2,3,4,5,6,7,8,9）**

**第二组数据：顺序表元素为 （1,1,2,2,2,3,4,5,5,5,6,6,7,7,8,8,9）**

**第三组数据：顺序表元素为 （1,2,3,4,5,5,6,7,8,8,9,9,9,9,9）**

* 算法设计与实现描述

（书上给出的基本运算外，其它问题必须先给出算法思想或步骤，再给出算法描述）

作业留过链表的，顺序表直接拿之前的函数改造一下即可，还是双指针，记录相同元素个数，然后批量移动删除

void DeWeightUpdate2(List \*L) {

int left = 0,right = 1,same = 0,count = 0;

while (right != L->count) {

if (L->data[left] == L->data[right]) {

same++;

} else {

if (same > 0) {

// 批量右移，减少移动次数

for (int i = right; i < L->count; i++) {

L->data[i - same] = L->data[i];

}

count++;

L->count -= same; // 更新元素个数

right -= same; // 调整 right

same = 0;

}

left = right;

}

right++;

}

// 处理末尾情况

if (same > 0) {

L->count -= same;

}

printf("Runned %d times\n",count);

}

* 运行结果截图及说明

测试代码与运行结果如下



4 顺序表扩展实验\*

（非必做内容，有兴趣的同学选做）

**1)（递增有序）顺序表表示集合A、B，实现：**

**C=A∪B，C=A∩B，C=A-B**

**A=A∪B，A=A∩B，A=A-B**

* 算法设计与实现描述

（书上给出的基本运算外，其它问题必须先给出算法思想或步骤，再给出算法描述）

//列表递增 设计算法实现集合 并、交、差、子集

// C = A 交 B

List JointSet(List list1,List list2) {

int ptr1 = 0,ptr2 = 0;

List new;

Init(&new);

while (ptr1 != list1.count && ptr2 != list2.count) {

int data1 = list1.data[ptr1];

int data2 = list2.data[ptr2];

if(data1 > data2) {

ptr2++;

} else if(data1 < data2) {

ptr1++;

} else {

new.data[new.count] = data1;

new.count++;

ptr1++;

ptr2++;

}

}

return new;

}

// C = A 并 B

List LinkSet(List list1, List list2) {

int ptr1 = 0, ptr2 = 0;

List new;

Init(&new);

while (ptr1 != list1.count && ptr2 != list2.count) {

int data1 = list1.data[ptr1];

int data2 = list2.data[ptr2];

if (data1 < data2) {

new.data[new.count++] = data1;

ptr1++;

} else if (data1 > data2) {

new.data[new.count++] = data2;

ptr2++;

} else {

new.data[new.count++] = data1;

ptr1++;

ptr2++;

}

}

while (ptr1 != list1.count) {

new.data[new.count] = list1.data[ptr1];

ptr1++;

new.count++;

}

while (ptr2 != list2.count) {

new.data[new.count] = list2.data[ptr2];

ptr2++;

new.count++;

}

return new;

}

// C = A 差 B

List DifferenceSet(List list1, List list2) {

int ptr1 = 0, ptr2 = 0;

List new;

Init(&new);

while (ptr1 != list1.count && ptr2 != list2.count) {

int data1 = list1.data[ptr1];

int data2 = list2.data[ptr2];

if (data1 < data2) {

new.data[new.count] = data1;

new.count++;

ptr1++;

} else if (data1 > data2) {

ptr2++;

} else {

ptr1++;

ptr2++;

}

}

while (ptr1 != list1.count) {

new.data[new.count] = list1.data[ptr1];

new.count++;

ptr1++;

}

return new;

}

// A = A 并 B

void LinkSet2(List \*list1, List list2) {

int ptr1 = 0, ptr2 = 0, count = 0;

int array[list1->count];

while (ptr1 != list1->count && ptr2 != list2.count) {

int data1 = list1->data[ptr1];

int data2 = list2.data[ptr2];

if (data1 < data2) {

array[count] = data1;

count++;

ptr1++;

} else if (data1 > data2) {

ptr2++;

} else {

ptr1++;

ptr2++;

}

}

while (ptr1 != list1->count) {

array[count] = list1->data[ptr1];

count++;

ptr1++;

}

// 重新写入 list1

list1->count = count;

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

list1->data[i] = array[i];

}

}

// A = A 交 B

void JointSet2(List \*list1, List list2) {

int ptr1 = 0, ptr2 = 0, count = 0;

int array[list1->count];

while (ptr1 != list1->count && ptr2 != list2.count) {

int data1 = list1->data[ptr1];

int data2 = list2.data[ptr2];

if (data1 < data2) {

ptr1++;

} else if (data1 > data2) {

ptr2++;

} else {

array[count] = data1;

count++;

ptr1++;

ptr2++;

}

}

// 重新写入list1

list1->count = count;

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

list1->data[i] = array[i];

}

}

// A = A 差 B

void DifferentSet2(List \*list1,List list2) {

int ptr1 = 0,ptr2 = 0,count = 0;

int array[list1->count];

while (ptr1 != list1->count && ptr2 != list2.count) {

int data1 = list1->data[ptr1];

int data2 = list2.data[ptr2];

if(data1 > data2) {

ptr2++;

} else if(data1 < data2) {

ptr1++;

} else {

Delete(list1,ptr1+1);

ptr2++;

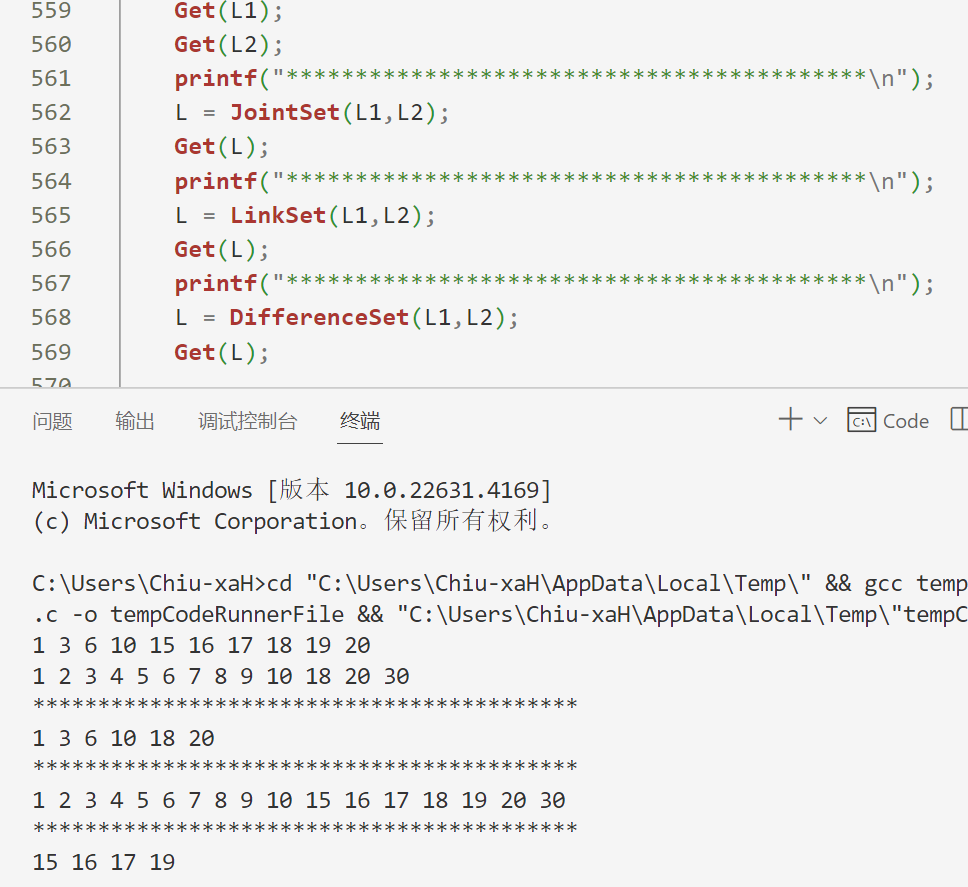
}

}

}

* 运行结果截图及说明

L1与L2为数据源，测试代码与运行结果如下





**2) 一个长度为L（L≥1）的升序序列S，处在第 个位置的数称为S 的中位数。例如，若序列S1=(11, 13, 15, 17, 19)，则S1 的中位数是15。两个序列的中位数是含它们所有元素的升序序列的中位数。例如，若S2=(2, 4, 6, 8, 20)，则S1 和S2 的中位数是11。**

**现有两个等长升序序列A 和B，试设计一个在时间和空间两方面都尽可能高效的算法，找出两个序列A 和B 的中位数。要求：**

**（1）给出算法的基本设计思想。**

**（2）根据设计思想，采用C 或C++语言描述算法，关键之处给出注释。**

**（3）说明你所设计算法的时间复杂度和空间复杂度。**

用合并函数，然后取中间值即可，合并函数可以直接拿上一道题的并集函数， 求有序顺序表中位数的函数需要自己设计

* 算法设计与实现描述

（书上给出的基本运算外，其它问题必须先给出算法思想或步骤，再给出算法描述）

// 求有序顺序表中位数

// C = A 并 B

List LinkSet(List list1, List list2) {

int ptr1 = 0, ptr2 = 0;

List new;

Init(&new);

while (ptr1 != list1.count && ptr2 != list2.count) {

int data1 = list1.data[ptr1];

int data2 = list2.data[ptr2];

if (data1 < data2) {

new.data[new.count++] = data1;

ptr1++;

} else if (data1 > data2) {

new.data[new.count++] = data2;

ptr2++;

} else {

new.data[new.count++] = data1;

ptr1++;

ptr2++;

}

}

while (ptr1 != list1.count) {

new.data[new.count] = list1.data[ptr1];

ptr1++;

new.count++;

}

while (ptr2 != list2.count) {

new.data[new.count] = list2.data[ptr2];

ptr2++;

new.count++;

}

return new;

}

// 求有序顺序表中位数

float Middle(List L) {

// 判断数量奇偶

if(L.count % 2 == 0) {

// 偶数 第n/2个数与第 (n/2)+1个数的平均值

return (L.data[L.count/2 - 1] + L.data[L.count/2]) / 2.0;

} else {

return L.data[L.count/2];

}

}

* 运行结果截图及说明

测试代码与运行结果如下



7总结、心得和建议

顺序表相比链表有优点也有劣势。通过对顺序表的学习和实战，能够对高级语言中的数组等类型有更好的理解，在实际应用中应尽量优化时间复杂度。