|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 |

实验二：链表实验

1实验目的

1. 熟练掌握线性表的链式存储结构。
2. 熟练掌握单链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的链式存储结构，并设计相关算法。

2 实验要求

1. 本次实验中的链表结构指带头结点的单链表；
2. 单链表结构和运算定义，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：linkedList.h
3. 程序运行、测试正确；
4. 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；
5. 程序有适当的注释，程序的书写要采用缩进格式。

3 实验任务

**编写算法实现下列问题的求解。**

1. **将单链表Ｌ中的奇数项和偶数项结点分解开（元素值为奇数、偶数），分别放入新的单链表中，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：单链表元素为 （1,2,3,4,5,6,7,8,9,10,20,30,40,50,60）**

**第二组数据：单链表元素为 （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）

还是用取余运算判断奇偶，与上个实验大同小异

//尾插法

Status TailAdd(LinkList \*L,Elemtype E) {

LinkList new = (LinkList)malloc(sizeof(Node));

LinkList list = (\*L);

while(list->next != NULL) {

list = list->next;

}

new->data = E;

new->next = NULL;

list->next = new;

return OK;

}

//设计算法分解链表为奇数链表、偶数链表

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

LinkList list = L->next;

Init(JiL);

Init(OuL);

while(list) {

if(list->data % 2 == 0) {

TailAdd(OuL,list->data);

} else {

TailAdd(JiL,list->data);

}

list = list->next;

}

}

* 运行结果截图及说明



1. **求两个递增有序单链表L1和L2中的公共元素，放入新的单链表L3中。**

**实验测试数据基本要求：**

**第一组**

**第一个单链表元素为 （1，3，6，10，15，16，17，18，19，20）**

**第二个单链表元素为 （1，2，3，4，5，6，7，8，9，10，18，20，30）**

**第二组**

**第一个单链表元素为 （1，3，6，10，15，16，17，18，19，20）**

**第二个单链表元素为 （2，4，5，7，8，9，12，22）**

**第三组**

**第一个单链表元素为 （）**

**第二个单链表元素为 （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）

就是递增集合求交集：

LinkList JointSetUpdate(LinkList L1,LinkList L2) {

LinkList list;

Init(&list);

LinkList list1 = L1->next;

while(list1) {

LinkList list2 = L2->next;

while(list2) {

if(list2->data == list1->data) {

TailAdd(&list,list2->data);

}

list2 = list2->next;

}

list1 = list1->next;

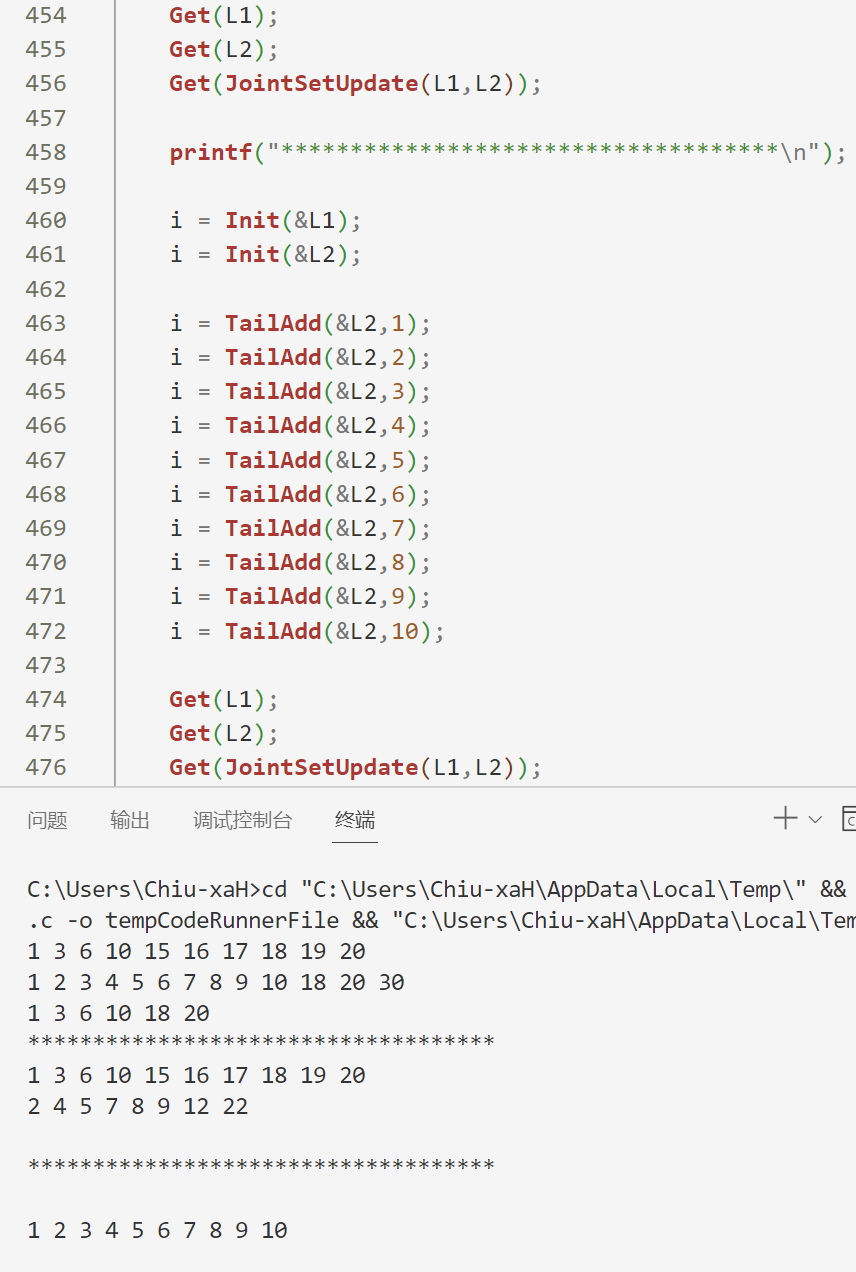
}

return list;

}

* 运行结果截图及说明

第二、三组没输出，因为没交集



1. 删除递增有序单链表中的重复元素，要求时间性能最好。

实验测试数据基本要求：

第一组数据：单链表元素为 （1,2,3,4,5,6,7,8,9）

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

第三组数据：单链表元素为 （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）

两个方法，时间是一致的，因为要释放free()每个结点，不得不逐个遍历，像顺序表可以批量移动，用双指针法就会得到明显提升。

//删除递增有序单链表中的重复元素，要求时间性能最好

//思路：用两个指针从头扫描，遇到相同的第一个指针固定，第二个继续直到不相同，然后删除中间重复的若干节点即可

void DeWeightUpdate(LinkList \*L) {

if (\*L == NULL || (\*L)->next == NULL)

return;

LinkList left = (\*L)->next;

while (left) {

LinkList right = left->next;

// 右指针没到尾 且 右指针和左指针重复值 则开始固定左指针，移动右指针直到不重复

while (right && right->data == left->data) {

LinkList temp = right;

right = right->next;

free(temp);

}

// 指向第一次非重复元素

left->next = right;

// 继续

left = right;

}

}

void DeWeight(LinkList \*L) {

if (\*L == NULL || (\*L)->next == NULL)

return;

LinkList list = (\*L)->next;

while (list && list->next) {

if (list->data == list->next->data) {

// 删除

LinkList old = list->next;

list->next = old->next;

free(old);

} else {

list = list->next;

}

}

}

* 运行结果截图及说明



1. **递增有序单链表L1、L2，不申请新结点，利用原表结点对两表进行合并，并使得合并后成为一个集合，合并后用L1的头结点作为头结点，删除多余的结点，删除L2的头结点。要求时间性能最好。实验测试数据基本要求：**

**第一组**

**第一个单链表元素为 （1，3，6，10，15，16，17，18，19，20）**

**第二个单链表元素为 （1，2，3，4，5，6，7，8，9，10，18，20，30）**

**第二组**

**第一个单链表元素为 （1，3，6，10，15，16，17，18，19，20）**

**第二个单链表元素为 （2，4，5，7，8，9，12，22）**

**第三组**

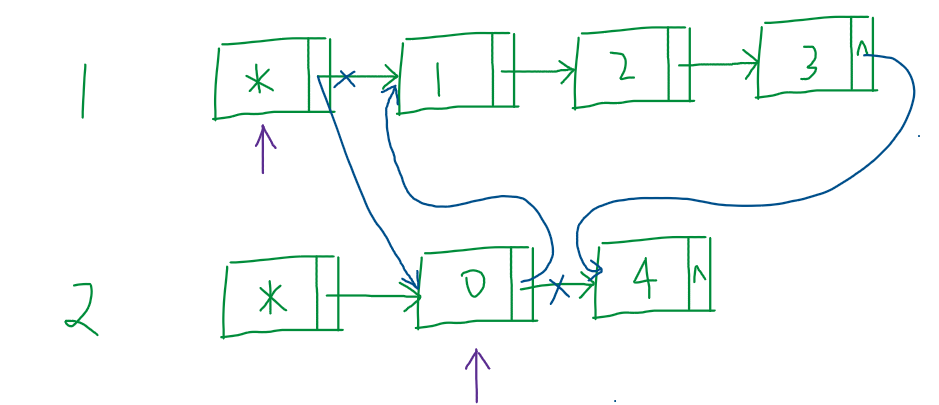
**第一个单链表元素为 （）**

**第二个单链表元素为 （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）

思路见注释：



// 递增有序单链表，原地合并，并删掉废弃的头结点

void Merge(LinkList \*L1, LinkList \*L2) {

if (\*L1 == NULL || \*L2 == NULL)

return;

LinkList list1 = \*L1, list2 = (\*L2)->next;

LinkList list1Next = NULL, list2Next = NULL;

while(list1->next && list2) {

if(list1->next->data >= list2->data) {

// 进行合并 2指针的结点插入到1指针的前面

// 记录原指针的下一个结点

list2Next = list2->next;

list1Next = list1->next;

// 修改2指针的next

list2->next = list1Next;

// 修改1指针的next

list1->next = list2;

// 完成修改 移动双指针

list2 = list2Next;

}

// 继续走1指针 不用合并

list1 = list1->next;

}

// 连接剩余部分

if(list2) {

list1->next = list2;

}

// 删除2的头结点

free(\*L2);

\*L2 = NULL;

}

* 运行结果截图及说明



1. **已知一个带有表头结点的单链表，结点结构如下图。假设该链表只给出了头指针list。在不改变链表的前提下，请设计一个尽可能高效的算法，查找链表中倒数第k个位置上的结点（k为正整数）。若查找成功，算法输出该结点的data值，并返回1；否则，只返回0。要求：**

**（1）描述算法的基本设计思想**

**（2）描述算法的详细实现步骤**

**（3）根据设计思想和实现步骤，采用程序设计语言描述算法（使用C 或C++语言实现），关键之处请给出简要注释。**

* 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）

思路如注释：

// 查找链表中倒数第k个位置上的结点（k为正整数）。若查找成功，算法输出该结点的data值，并返回1；否则，只返回0。

Status FindK(LinkList L, int k) {

if(k <= 0) {

return ERROR;

}

LinkList right = L->next, left = L->next;

int count = 0;

// 右指针先走k步

while (right && count < k) {

right = right->next;

count++;

}

// 验证k合法 如果提前走到头了，说明k不合法

if (count < k) {

return ERROR;

}

// 双指针同时移动，直到右指针走到末尾

while (right) {

right = right->next;

left = left->next;

}

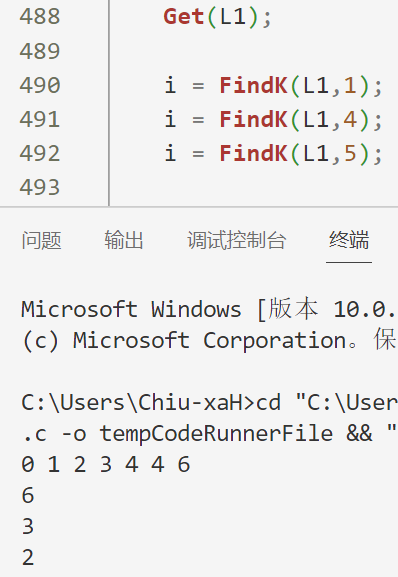
// 左结点的值

printf("%d\n", left->data);

return OK;

}

* 运行结果截图及说明



6总结、心得和建议

链表的确相比顺序表解决了一些劣势，但它并未十全十美，具体使用需要结合自己的需求选择。