**第2章 线性表作业**

**一、填空**

**1. 在顺序表中插入或删除一个元素，需要平均移动 \_表中一半\_元素，具体移动的元素个数**

**与 顺序表的长度和插入或删除元素在表中的位置 有关。**

**2. 线性表中结点的集合是 有限 的，结点间的关系是 \_\_\_\_一对一\_\_ 的。**

**3. 向一个长度为n的向量的第i个元素(1≤i≤n+1)之前插入一个元素时，需向后移动 的元素个数为 n - i + 1 。**

**4. 向一个长度为n的向量中删除第i个元素(1≤i≤n)时，需向前移动\_\_\_n - i\_\_\_ 个元素。**

**5. 在顺序表中访问任意一结点的时间复杂度均为 O(1) ，因此，顺序表也称为\_\_随机存取\_\_\_\_的数据结构。**

**6. 顺序表中逻辑上相邻的元素的物理位置 一定 相邻。单链表中逻辑上相邻的元素的物理位置 不一定 相邻。**

**7. 在单链表中，除了首元结点外，任一结点的存储位置由\_\_\_\_ 直接前驱结点链域的值 \_\_\_\_\_指示。**

**8． 在n个结点的单链表中要删除已知结点\*p，需找到它的\_直接前驱结点的地址\_\_\_\_\_ ，其时间复杂度为 O(n) 。**

1. **已知P为单链表中的非首尾结点，在P结点后插入S结点的语句为： s->next = p->next; \_\_\_p->next= s ;\_\_\_\_\_\_\_\_\_\_\_\_\_ 。  
   10.在非空双向循环链表中，在结点q的前面插入结点p的过程如下：   
   p->prior=q->prior;  
   q->prior->next=p;  
   p->next=q;  
   q->prior = p\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;**

**二、判断正误**

**（ × ）1. 链表的每个结点中都恰好包含一个指针。**

**错误。链表中的每个结点可含多个指针域，分别存放多个指针。例如双向链表中的结点可以含有两个指针域，分别存放指向其直接前趋和直接后继的指针。**

**（ × ）2. 链表的物理存储结构具有同链表一样的顺序。**

**错误。在链表中逻辑相邻的元素的物理位置不一定相邻。**

**（ × ）3. 链表的删除算法很简单，因为当删除链中某个结点后，计算机会自动将后续各个单元向前移动。**

**错误。在链表中进行插入和删除操作时，后续各个单元的移动需要程序控制。**

**（ × ）4. 线性表的每个结点只能是一个简单类型，而链表的每个结点可以是一个复杂类型。**

**错误。线性表的每个结点也可以是一个复杂的类型。比如用“结构体”做为线性表的节点**

**也是可以的。而结构体是复杂类型。**

**（ × ）5. 顺序表结构适宜于进行顺序存取，而链表适宜于进行随机存取。**

**错误。顺序表结构适宜于进行随机存取而链表适宜于进行顺序存取。**

**（ × ）6. 顺序存储方式的优点是存储密度大，且插入、删除运算效率高。**

**错误。顺序存储方式的插入、删除运算效率并不高。**

**（ × ）7. 线性表在物理存储空间中也一定是连续的。**

**错误。当线性表以链表的形式被存储时，它在物理空间中就不一定是连续的。**

**（ × ）8. 线性表在顺序存储时，逻辑上相邻的元素未必在存储的物理位置次序上相邻。**

**错误。当线性表在顺序存储时，逻辑上相邻的元素一定在存储的物理位置次序上相邻。**

**（ × ）9. 顺序存储方式只能用于存储线性结构。**

**错误。顺序存储方式不仅能用于存储线性结构，还可以用来存放非线性结构，像完全二**

**叉树就是非线性结构，但它最好的存储方式是顺序存储方式。**

**（ × ）10. 线性表的逻辑顺序与存储顺序总是一致的。**

**错误。顺序存储方式总是一致的，而链式的则不一定。**

**三、单项选择题**

**（ C ）1．数据在计算机存储器内表示时，物理地址与逻辑地址相同并且是连续的，称之为：**

**（A）存储结构 （B）逻辑结构 （C）顺序存储结构 （D）链式存储结构**

**（ B ）2. 一个向量第一个元素的存储地址是100，每个元素的长度为2，则第5个元素的地址是**

**（A）110 （B）108 （C）100 （D）120**

**解析：地址=100+2 \* (5 - 1) ;**

**（ A ）3. 在n个结点的顺序表中，算法的时间复杂度是O（1）的操作是：**

1. **访问第i个结点（1≤i≤n）和求第i个结点的直接前驱（2≤i≤n）**
2. **在第i个结点后插入一个新结点（1≤i≤n）**
3. **删除第i个结点（1≤i≤n） （D） 将n个结点从小到大排序**

**解析：在顺序结构中，随机访问的时间复杂度是O(1)；**

**（ B ）4. 向一个有127个元素的顺序表中插入一个新元素并保持原来顺序不变，平均要移动 个元素**

**（A）8 （B）63.5 （C）63 （D）7**

**解析：平均要移动线性表一半长度的元素；**

**（ A ）5. 链接存储的存储结构所占存储空间：**

1. **分两部分，一部分存放结点值，另一部分存放表示结点间关系的指针**
2. **只有一部分，存放结点值**

**（C） 只有一部分，存储表示结点间关系的指针**

**（D） 分两部分，一部分存放结点值，另一部分存放结点所占单元数**

**（ B ）6. 链表是一种采用 存储结构存储的线性表；**

**（A）顺序 （B）链式 （C）星式 （D）网状**

**（ D ）7. 线性表若采用链式存储结构时，要求内存中可用存储单元的地址:**

**（A）必须是连续的 （B）部分地址必须是连续的**

**（C）一定是不连续的 （D）连续或不连续都可以**

**（ B ）8． 线性表Ｌ在 情况下适用于使用链式结构实现。**

**（Ａ）需经常修改Ｌ中的结点值 （Ｂ）需不断对Ｌ进行删除插入**

**（Ｃ）Ｌ中含有大量的结点 （Ｄ）Ｌ中结点结构复杂**

**解析：当需要不断的进行删除插入操作时，考虑使用链式结构；**

**（ C ）9． 单链表的存储密度**

**（Ａ）大于1； （Ｂ）等于1； （Ｃ）小于1； （Ｄ）不能确定**

**解析：存储密度=数据元素本身所占的存储量和整个结点结构所占的存储量之比；**

**（ B ）10． 设a1、a2、a3为3个结点，整数P0，3，4代表地址，则如下的链式存储结构称为**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | P0 |  |  | **3** |  |  | **4** |  |  |
| **P0** | **🡪** | **a1** | **3** | **🡪** | **a2** | **4** | **🡪** | **A3** | **0** |

**（Ａ）循环链表 （Ｂ）单链表 （Ｃ）双向循环链表 （Ｄ）双向链表**

**四、算法题**

1. **假设两个按元素值递增有序排列的线性表A和B，均以单链表作为存储结构，请编写算法，将A表和B表归并成一个按元素值递减有序排列的线性表C，并要求利用原表（即A表和B表的）结点空间存放表C。**

**[解答]：**

**Typedef struct Node{**

**Char data ;**

**Struct node\* next ;**

**}node, \*Linklist ;**

**Linklist CreateList() {**

**Linklist l ;**

**Char c ;**

**L = (linklist)malloc(size(node))//由系统生成一个lnode型的结点，同时将该结点的起始位置赋给指针变量l；**

**l->next = null ;**

**Node \*p ;**

**Tail = l ;**

**C = getchar() ;**

**While(c!=EOF){  
p = (\*node) mallloc(sizeof(node)) ;**

**p->data = c ;**

**P->next = l->next ;**

**l->next = p ;//将p插入到l的后面**

**C = getchar() ;**

**}**

**Return l ;**

**}**

**Void ReverseList(LinkList L){**

**Node \*p, \*q ;**

**P = l->next ;**

1. **>next = null ;**

**While(p != null) {**

**q= p->next ;**

1. **>next = l->next ;**

**l->next = p ;**

**P = q ;**

**}**

**}**

**Void MergeList(List la, List lb){**

**//已知线性表la和lb中的数据元素按值递增排列**

**//归并la和lb得到表lc，且lc按值递减排列**

**ReverseList(la) ; //转置la，使线性表la数值按值递减排列**

**ReverseList(lb) ; //同理，转置lb**

**LinkList lc;**

**Node \*pa, \*pb, \*pc ;**

**pa = la->next ;**

**pb = lb->next ;**

**lc = pc = la ; //用la的头结点作为lc的头结点；**

**lc->next = null ;**

**While(pa != null && pb != null) {**

**If(pa ->data <= pb ->data) {**

1. **>next = pb ;**

**Pc = pb ;**

**Pb = pb->next; //遍历lb中存储的元素;}**

**Else{**

1. **>next = pa ;**

**Pc = pa ;**

**Pa = pa->next; //遍历lb中存储的元素;}}**

**Pc->next = pa ? pa : pb ; //插入剩余段；**

**}**

**Int main(){**

**Linklist la ;**

**La = CreatList();**

**Getchar();**

**Linklist lb ;**

**Lb = creatlist();**

**Linklist lc ;**

**Lc = MergeList(la,lb) ;**

**Return 0 ;}**

**2、 设线性表A=(a1, a2,…,am)，B=(b1, b2,…,bn)，试写一个按下列规则合并A、B为线性表C的算法，使得：**

**C= (a1, b1,…,am, bm, bm+1, …,bn) 当m≤n时；**

**或者 C= (a1, b1,…,an, bn, an+1, …,am) 当m>n时。**

**线性表A、B、C均以双链表作为存储结构，且C表利用A表和B表中的结点空间构成。**

**[解答]：**

**Typedef struct lNode{**

**Int data ;**

**Lnode\* next ;**

**Lnode\* prior ;**

**}lnode,\*dulinklist ;**

**//初始化双链表**

**void initlist(dulinklist &l){**

**L = (dulinklist)malloc(sizeof(lnode)) ;**

**l->next = null ;**

**L->prior = null ;**

**}**

**//将新元素e放入链表中**

**void creatlist(dulinklist &l,int e){**

**duLinklist p, temp ;**

**P = l ;**

**temp = (dulinklist)malloc(sizeof(lnode));**

**While(p->next)**

**p = p->next ;**

**temp->data = e ;**

**temp->next = null ;**

**p->next = temp;**

**}**

**Void MergeList(List &la, List &lb, List &lc){**

**// 已知la和lb都是双向链表，要实现算法，只需将lb间隔插入la即可；**

**Node \*pa, \*pb, \*qa, \*qb;**

**pa = la->next ; //初始化，pa指向第一个结点**

**pb = lb->next ;**

**Lc = la ; //因为lc的头结点就是la，只需将lc指向la即可;**

**while(pa && pb){//当pa和pb都非空时，进入下述循环**

**Pa = qa ;**

**Pb = qb ;**

**Pa = pa->next ;**

**Pb = pb->next ;**

**//在对结点进行操作前就先保存下下一个结点的地址，这样可以避免在后续处理更复杂的指针问题;**

**//将qb插入到qa的前面;**

**qb->prior = qa -> prior ;**

**qa->prior->next = qb;**

**qb->next = qa ;**

**qa->prior = qb ;**

**}**

**If(!pa)//如果la的长度小于lb**

**{qb->next = pb ;//直接把lb剩余的段接到新段后面**

**}**

**pb = lb ;**

**free(pb)；删除B链表;**

**}**

**Int main(){**

**Linlist la, lb, lc;**

**Initlist(la) ;**

**Initlist(lb) ;**

**Initlist(lc) ;**

**For(int i = 0; i < m; i ++)**

**Creatlist(la,i ); //向la中添加元素**

**}**

**For(int j = 0; j < n; j ++)**

**Creatlist(lb,j ); //向lb中添加元素**

**}**

**Listmerge(la, lb, lc) ;//合并链表；**

**Return 0;**

**}**