得分	

一、单项选择题（X 题，每题 X 分，共 X 分）

评分标准：每题回答正确得 1.5 分，错误不得分！

1、数据结构中的线性表是（ **A** ）。

A. 一个有限序列，可以为空　　B. 一个有限序列，不可以为空

C. 一个无限序列，可以为空　　D. 一个无限序列，不可以为空

2、在一个长度为 n 的顺序表中删除第 i 个元素（1<=i<=n）时，需向前移动（ **A** ）个元素。

A. n-i　　B. n-i+l　　C. n-i-1　　D. i

3、在一个长度为 n 的顺序表中删除下标为 i 的元素（0<=i<n）时，需向前移动（ **C** ）个元素。

A. n-i　　B. n-i+l　　C. n-i-1　　D. i

4、线性表采用链式存储时，其地址（ **D** ）。

A. 必须是连续的　　B. 一定是不连续的

C. 部分地址必须是连续的　　D. 连续与否均可以

5、从一个具有 n 个结点的单链表中查找其值等于 x 的结点时，在查找成功的情况下，需平均比较（ **C** ）个元素结点。

A. n/2　　B. n　　C. (n+1)/2　　D. (n-1)/2

6、在双向循环链表中，在 p 所指的结点之后插入 s 指针所指的结点，其操作是（ **D** ）。

A. p->next=s;　s->prior=p;
p->next->prior=s; s->next=p->next;

B. s->prior=p;　s->next=p->next;
p->next=s;　p->next->prior=s;

C. p->next=s;　p->next->prior=s;
s->prior=p;　s->next=p->next;

D. s->prior=p;　s->next=p->next;
p->next->prior=s;　p->next=s;

7、设单链表中指针 p 指向结点 m，若要删除 m 之后的结点（若存在），则需修改指针的操作为（ **A** ）。

A. p->next=p->next->next;　　B. p=p->next;

C. p=p->next->next;　　D. p->next=p;

8、在一个长度为 n 的顺序表中向第 i 个元素(0< i<n+l)之前插入一个新元素时，需向后移动（ **B** ）个元素。

A. n-i　　B. n-i+l　　C. n-i-1　　D. i

9、在一个单链表中，已知 q 结点是 p 结点的前趋结点，若在 q 和 p 之间插入 s 结点，则须执行（ **B** ）。

A. s->next=p->next;　p->next=s　　B. q->next=s;　s->next=p

C. p->next=s->next;　s->next=p　　D. p->next=s;　s->next=q

10、以下关于线性表的说法不正确的是（ **C** ）。

A. 线性表中的数据元素可以是数字、字符、记录等不同类型。

B. 线性表中包含的数据元素个数不是任意的。

C. 线性表中的每个结点都有且只有一个直接前趋和直接后继。

D. 存在这样的线性表：表中各结点都没有直接前趋和直接后继。

11、线性表的顺序存储结构是一种（ **A** ）的存储结构。

A. 随机存取　　B. 顺序存取　　C. 索引存取　　D. 散列存取

12、在顺序表中，只要知道（ **D** ），就可在相同时间内求出任一结点的存储地址。

A. 基地址　　B. 结点大小　　C. 向量大小　　D. 基地址和结点大小

13、在等概率情况下，顺序表的插入操作要移动（ **B** ）结点。

A. 全部　　B. 一半　　C. 三分之一　　D. 四分之一

14、在（ **C** ）运算中，使用顺序表比链表好。

A. 插入　　B. 删除

C. 根据序号查找　　D. 根据元素值查找

15、在一个具有 n 个结点的有序单链表中插入一个新结点并保持该表有序的时间复杂度是（ **B** ）。

A. O(1)　　B. O(n)　　C. $O(n^2)$　　D. $O(\log_2 n)$

16、在带有头结点的单链表 HL 中，要向表头插入一个由指针 p 指向的结点，则执行（ **A** ）。

A. p->next=HL->next; HL->next=p;　　B. p->next=HL; HL=p;

C. p->next=HL; p=HL;　　D. HL=p; p->next=HL;

17、对线性表，在下列哪种情况下应当采用链表表示？（ **B** ）。

A. 经常需要随机地存取元素　　B. 经常需要进行插入和删除操作

C. 表中元素需要占据一片连续的存储空间　　D. 表中元素的个数不变

18、在一个单链表中，若 q 所指结点是 p 所指结点的前驱结点，若在 q 与 p 之间插入一个 s 所指的结点，则执行（ **D** ）。

A. s->link=p->link;　p->link=s;　　B. p->link=s;　s->link=q;

C. p->link=s->link;　s->link=p;　　D. q->link=s;　s->link=p;

19、线性表采用链式存储时，结点的存储地址（ **B** ）。

A. 必须是不连续的　　B. 连续与否均可

C. 必须是连续的　　D. 和头结点的存储地址相连续

20、将长度为 n 的单链表链接在长度为 m 的单链表之后的算法的时间复杂度为（ **C** ）。

A. O(1)　　B. O(n)　　C. O(m)　　D. O(m+n)

21、设指针变量 p 指向单链表结点 A，则删除结点 A 的后继结点 B 需要的操作为（ **A** ）。

A. p->next=p->next->next　　B. p=p->next

C. p=p->next->next　　D. p->next=p

22、下面关于线性表的叙述错误的是（ **D** ）。

A. 线性表采用顺序存储必须占用一片连续的存储空间

B. 线性表采用链式存储不必占用一片连续的存储空间

C. 线性表采用链式存储便于插入和删除操作的实现

D. 线性表采用顺序存储便于插入和删除操作的实现

23、设指针变量 p 指向单链表中结点 A，若删除单链表中结点 A，则需要修改指针的操作序列为（ **A** ）。

A. q=p->next;　p->data=q->data;　p->next=q->next;　free(q);

B. q=p->next;　q->data=p->data;　p->next=q->next;　free(q);

C. q=p->next;　p->next=q->next;　free(q);

D. q=p->next;　p->data=q->data;　free(q);

24、设一维数组中有 n 个数组元素，则读取第 i 个数组元素的平均时间复杂度为（ **C** ）。

A. O(n)　　B. O($n\log_2 n$)　　C. O(1)　　D. O(n^2)

25、设一个有序的单链表中有 n 个结点，现要求插入一个新结点后使得单链表仍然保持有序，则该操作的时间复杂度为（ **D** ）。

A. O($\log_2 n$)　　B. O(1)　　C. O(n^2)　　D. O(n)

26、设一条单链表的头指针变量为 head 且该链表没有头结点，则其判空条件是（ **A** ）。

A. head==0　　B. head->next==0

C. head->next==head　　D. head!=0

27、设带有头结点的单向循环链表的头指针变量为 head，则其判空条件是（ **C** ）。

A. head==0　　B. head->next==0

C. head->next==head　　D. head!=0

28、设顺序线性表中有 n 个数据元素，则删除表中第 i 个元素需要移动（ **A** ）个元素。

A. n-i　　B. n+l -i　　C. n-1-i　　D. i

29、设指针变量 p 指向双向链表中结点 A，指针变量 s 指向被插入的结点 X，则在结点 A 的后

面插入结点 X 的操作序列为（ **D** ）。

A. p->right=s;　s->left=p;　p->right->left=s;　s->right=p->right;

B. s->left=p;　s->right=p->right;　p->right=s;　p->right->left=s;

C. p->right=s;　p->right->left=s;　s->left=p;　s->right=p->right;

D. s->left=p;　s->right=p->right;　p->right->left=s;　p->right=s;

30、设某链表中最常用的操作是在链表的尾部插入或删除元素，则选用下列（ **D** ）存储方式最节省运算时间。

A. 单向链表　　B. 单向循环链表　　C. 双向链表　　D. 双向循环链表

31、设指针 q 指向单链表中结点 A，指针 p 指向单链表中结点 A 的后继结点 B，指针 s 指向被插入的结点 X，则在结点 A 和结点 B 插入结点 X 的操作序列为（ **B** ）。

A. s->next=p->next; p->next=-s;　　B. q->next=s; s->next=p;

C. p->next=s->next; s->next=p;　　D. p->next=s; s->next=q;

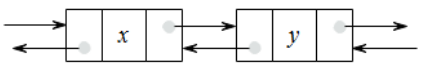
得分

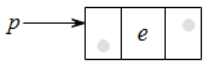
二、填空题（5 题，每空 1 分，共 15 分）

评分标准：每空回答正确得 1 分，错误不得分，不完全正确则酌情给分！

1、线性表（$a_1,a_2,\cdots,a_n$）的顺序存储结构中，设每个单元的长度为 L，元素 a_i 的存储地址 LOC(a_i) 为 **【1】LOC(a_1)+(i-1)*L** 。

2、已知一双向链表如下(指针域名为 next 和 prior)。现将 p 所指的结点插入到 x 和 y 结点之间，其操作步骤为：**【2】p->next=y（或 p->next=x->next）**；**【3】x->next->prior=p（或 y->prior=p）**；**【4】x->next=p**；**【5】p->prior=x**；





3、线性结构反映结点间的逻辑关系是**【6】一对一**的，非线性结构反映结点间的逻辑关系是**【7】一对多或多对多**的。

4、线性表是一种典型的**【8】线性**结构。

5、在一个长度为 n 的顺序表的第 i 个元素之前插入一个元素，需要后移**【9】n-i+1**个元素。

6、顺序表中逻辑上相邻的元素的物理位置**【10】相邻**。

7、要从一个顺序表删除一个元素时，被删除元素之后的所有元素均需**【11】前移**一个位置，移动过程是从**【12】前**（前/后）向**【13】后**（前/后）依次移动每一个

元素。

8、在线性表的顺序存储中，元素之间的逻辑关系是通过【14】物理存储位置决定的；在线性表的链接存储中，元素之间的逻辑关系是通过【15】链域的指针值决定的。

9、在双向链表中，每个结点含有两个指针域，一个指向【16】前趋结点，另一个指向【17】后继结点。

10、当对一个线性表经常进行存取操作，而很少进行插入和删除操作时，则采用【18】顺序存储结构为宜。相反，当经常进行的是插入和删除操作时，则采用【19】链接（或链式）存储结构为宜。

11、顺序表中逻辑上相邻的元素，物理位置【20】一定（一定/不一定）相邻，单链表中逻辑上相邻的元素，物理位置【21】不一定（一定/不一定）相邻。

12、线性表、栈和队列都是【22】线性结构，可以在线性表的【23】任何位置插入和删除元素；对于栈只能在【24】栈顶位置插入和删除元素；对于队列只能在【25】队尾位置插入元素和在【26】队头位置删除元素。

13、根据线性表的链式存储结构中每个结点所含指针的个数，链表可分为【27】单链表和【28】双链表；而根据指针的联接方式，链表又可分为【29】非循环链表和【30】循环链表。

14、在单链表中设置头结点的作用是【31】使空表和非空表统一、算法处理一致。

15、对于一个具有 n 个结点的单链表，在已知的结点 p 后插入一个新结点的时间复杂度为【32】O(1)，在给定值为 x 的结点后插入一个新结点的时间复杂度为【33】O(n)。

16、对于一个长度为 n 的单链存储的线性表，在表头插入元素的时间复杂度为【34】O(1)，在表尾插入元素的时间复杂度为【35】O(n)。

17、在下面的数组 a 中链接存储着一个线性表，表头指针为 a[0].next，则该线性表为【36】（38，56，25，60，42，74）。

		0	1	2	3	4	5	6	7	8
a	data		60	56	42	38		74	25	
	next	4	3	7	6	2		0	1	

18、在以 HL 为表头指针的带表头附加结点的单链表和循环单链表中，判断链表为空的条件分别为【37】HL->next == NULL和【38】HL->next == HL（设结点的指针域为 next）。

19、在一个带头结点的单循环链表中，p 指向尾结点的直接前驱，则指向头结点的指针 head 可用 p 表示为 head=【39】p->next->next（设结点的指针域为 next）。

20、设指针变量 p 指向单链表中结点 A，指针变量 s 指向被插入的新结点 X，则进行插入操作的语句序列为【40】s->next = p->next; p->next = s;（设结点的指针域为 next）。

21、设指针变量 p 指向单链表中结点 A，指针变量 s 指向被插入的结点 X，则在结点 A 的后面

插入结点 X 需要执行的语句序列：s->next=p->next; 【41】p->next = s ;（设结点的指针域为 next）。

22、设指针变量 p 指向双向循环链表中的结点 X，则删除结点 X 需要执行的语句序列为【42】p>llink->rlink=p->rlink; p->rlink->llink=p->llink;（设结点中的两个指针域分别为 llink 和 rlink）。

23、设顺序线性表中有 n 个数据元素，则第 i 个位置上插入一个数据元素需要移动表中【43】n+1-i 个数据元素；删除第 i 个位置上的数据元素需要移动表中【44】n-i 个元素。

24、设指针变量 p 指向双向链表中的结点 A，指针变量 s 指向被插入的结点 X，则在结点 A 的后面插入结点 X 的操作序列为【45】s->left =p; s->right=p->right; p->right->left=s; 【46】p->right =s;（设结点中的两个指针域分别为 left 和 right）。

25、设指针变量 head 指向双向链表中的头结点，指针变量 p 指向双向链表中的第一个结点，则指针变量 p 和指针变量 head 之间的关系是 p=【47】head->rlink 和 head=【48】p->llink（设结点中的两个指针域分别为 llink 和 rlink）。

26、设指针 p 指向单链表中结点 A，指针 s 指向被插入的结点 X，则在结点 A 的前面插入结点 X 时的操作序列为：1) s->next=【49】p->next ；2) p->next=s；3) t=p->data；4) p->data=【50】s->data ； 5) s->data=t；

27、设指针变量 p 指向单链表中结点 A，则删除结点 A 的语句序列为：q=p->next; p->data=q->data; p->next=【51】q->next ; feee(q);

28、下面程序段的功能是利用从尾部插入的方法建立单链表的算法，请在下划线处填上正确的内容。

```
typedef struct node {
    int data;
    struct node *next;
} lklist;

void lklistcreate(【52】lklist  *& head ) {
    for (i=1; i<=n; i++) {
        p=(lklist *)malloc(sizeof(lklist));
        scanf("%d", &(p->data));
        p->next=0;
        if(i==1) head=q=p;
        else {
            q->next=p;
            【53】q=p ;
        }
    }
}
```

得分	

三、判断题（X 题，每题 1X 分，共 X 分。正确填‘T’，错误“F”。）

评分标准：每题回答正确得 2 分，错误不得分！

1、线性表的顺序存储结构比链式存储结构更好。 （ **F** ）

2、线性表中的所有元素都有一个前驱元素和后继元素。 （ **F** ）

3、不论线性表采用顺序存储结构还是链式存储结构，删除值为 X 的结点的时间复杂度均为 O(*n*)。 （ **T** ）

4、非空的双向循环链表中任何结点的前驱指针均不为空。 （ **T** ）

5、对链表进行插入和删除操作时不必移动链表中结点。 （ **T** ）

得分	

四、阅读程序题（X 题，每空 X 分，共 X 分）

评分标准：每空回答正确得 1 分，错误不得分，不完全正确则酌情给分！

1、阅读程序回答问题。（评分标准：*第 1 点 4 分，第 2 点 1 分!*）

```
LinkList mynote(LinkList *&L) { //L 是不带头结点的单链表的头指针
    if(L&&L->next) {
        q=L;
        L=L->next;
        p=L;
    S1: while(p->next) p=p－>next;
    S2:     p->next=q; q->next=NULL;
    }
    return  L;
}
```

（1）说明语句 S1 的功能；

（2）说明语句组 S2 的功能；

（3）设链表表示的线性表为（$a_1,a_2,\cdots,a_n$）,写出算法执行后的返回值所表示的线性表。

答：（1）查询链表的尾结点；

（2）将第一个结点链接到链表的尾部，作为新的尾结点；（3）返回的线性表为（$a_2,a_3,\cdots,a_n,a_1$）。

2、阅读程序回答问题：（1）说明语句 S1 的功能；（2）说明语句组 S2 的功能；（3）设链表表示的线性表为（$a_1,a_2,\cdots,a_n$），写出算法执行后的返回值所表示的线性表。（评分标准：*第 1 点 4 分，第 2 点 1 分!*）

```
LinkList mynote(LinkList *&L) { //L 是不带头结点的单链表的头指针
    if(L&&L->next) {
        q=L;
```

```
        L=L->next;
        p=L;

    S1：while(p->next) p=p->next;
    S2：p->next=q;      q->next=NULL;
    }
    return   L;
}
```

答：（1）查询链表的尾结点；

（2）将第一个结点链接到链表的尾部，作为新的尾结点；

（3）返回的线性表为（$a_2, a_3, \cdots, a_n, a_1$）；

得分	五、应用题（X 题，每题 X 分，共 X 分）
	评分标准：每题回答完全正确得 5 分，其余按得分点给分！

1、设指针变量 p 指向双向链表中结点 A，指针变量 q 指向被插入结点 B，要求给出在结点 A 的后面插入结点 B 的操作序列（设双向链表中结点的两个指针域分别为 llink 和 rlink）。（评分标准：*共 5 分。第 1 点 4 分，第 2 点 1 分！*）

答：q->llink=p; q->rlink=p->rlink; p->rlink->llink=q; p->rlink=q;

得分	六、编程题（X 题，每题 X 分，共 X 分）
	评分标准：每小题回答正确得 10 分，不完全正确则按得分点给分。

1、已知线性表($a_1,a_2,\cdots,a_n$)采用顺序存储结构，其类型 SqList 的定义如下。设计一个算法，删除其元素值为 x 的结点（假若 x 是唯一的）。并求出其算法的平均时间复杂度。实现算法的函数原型为：Status ***ListDelete***(Sqlist *& L, Elemtype *x*)；（评分标准：*第 1 点 4 分，第 2 点 1 分！*）

```
#define   LIST_INIT_SIZE   100     //顺序表初始分配容量
typedef   struct   {
    Elemtype     *elem;            //顺序存储空间基址
    int   length;                  //当前长度（存储元素个数）
} SqList;
```

答：Status ***ListDelete***(Sqlist *& L, Elemtype *x*) {

```
int i,j;
for(i=0; i<L->length; i++)
    if(L->elem[i]==x) break;
if(i=L->length) return ERROR;
for(j=i; j<L->lengthi-1; j++)
    L->elem[j]=L->elem[j+1];
L->length--;
```

}　　（8 分）

设元素个数记为 n，在线性表 L 中删除数据元素概率为 P_i，不失一般性，设各个位置是等概率，则 $P_i=1/n$。

[a] 比较的平均次数: $E_{compare}=\sum p_i*i$　$(1\leqq i\leqq n)$。　∴ $E_{compare}=(n+1)/2$。

[b] 删除时平均移动次数: $E_{delete}=\sum p_i*(n-i)$　$(1\leqq i\leqq n)$。∴ $E_{delete}=(n-1)/2$。

所以，总体平均时间复杂度: $E_{compare}+E_{delete}=n$ ，即为 O(n)。　　（2 分）

2、设计在无头结点的单链表中删除第 i 个结点的算法。实现算法的函数原型为：void ***delete***(LinkList *& q, int i);（评分标准：*第 1 点 4 分，第 2 点 1 分!*）

答：（1）算法思想为：

A、应判断删除位置的合法性，当 i<0 或 i>n-1 时，不允许进行删除操作；

B、当 i=0 时，删除第一个结点：

C、当 0 <i<n 时，允许进行删除操作，但在查找被删除结点时，须用指针记住该结点的前趋结点。

（2）算法描述如下：

```
void delete(LinkList *& q, int i) {     //在无头结点的单链表中删除第 i 个结点
    LinkList   *p,*s;
    int   j;
    if(i<0)
        printf("Can't delete");
    else if(i= =0) {
        s=q;
        q=q->next;
        free(s);
    }else {
        j=0;
        s=q;
        while((j<i) && (s! = NULL)) {
            p=s;
            s=s->next;
            j++;
        }

        if(s= =NULL)
            printf("Cant't   delete");
        else{
            p->next=s->next;
            free(s);
        }
```

```
    }
}
```

3、在单链表上实现线性表的求表长 ListLength(L)运算。实现算法的函数原型为：int ***ListLength***(LinkList *& L)；（评分标准：*第 1 点 4 分，第 2 点 1 分!*）

答：（1）由于在单链表中只给出一个头指针，所以只能用遍历的方法来数单链表中的结点个数。

（2）算法描述如下：

```
int ListLength( LinkList *& L ) {   //求带头结点的单链表的表长
    int len=0;
    ListList *p;
    p=L;
    while( p->next!=NULL ) {
        p=p->next;
        len++;
    }
    return (len);
}
```

4、设计将带表头的链表逆置算法。实现算法的函数原型为：void ***invert***(LinkList *& head)；（评分标准：*第 1 点 4 分，第 2 点 1 分!*）

答：（1）设单循环链表的头指针为 head，类型为 LinkList。逆置时需将每一个结点的指针域作以修改，使其原前趋结点成为后继。如要更改 q 结点的指针域时，设 s 指向其原前趋结点，p 指向其原后继结点，则只需进行 q->next=s;操作即可。

（2）算法描述如下：

```
void invert(LinkList *& head) {
    //逆置 head 指针所指向的单循环链表
    linklist *p, *q, *s;
    q=head;
    p=head->next;
    while (p!=head) { //当表不为空时，逐个结点逆置
        s=q;
        q=p;
        p=p->next;
        q->next=s;
    }
    p->next=q;
}
```

5、假设有一个带表头结点的链表，表头指针为 head，每个结点含三个域：data, next 和 prior。其中 data 为整型数域，next 和 prior 均为指针域。现在所有结点已经由 next 域连接起来，试编一个算法，利用 prior 域（此域初值为 NULL）把所有结点按照其值从小到大的顺序链接起来。实现算法的函数原型为：void ***insert***(LinkList *& head)；（评分标准：*第 1 点 4 分，第*

2 点 1 分!）

答：（1）定义类型 LinkList 如下：

```
typedef struct node {
    int data;
    struct node *next,*prior;
} LinkList;
```

此题可采用插入排序的方法，设 p 指向待插入的结点，用 q 搜索已由 prior 域链接的有序表找到合适位置将 p 结点链入。

（2）算法描述如下：

```
void insert(LinkList *& head) {
    LinkList *p,*s,*q;
    p=head->next;   //p 指向待插入的结点，初始时指向第一个结点
    while(p!=NULL) {
        s=head;   // s 指向 q 结点的前趋结点
        q=head->prior;   //q 指向由 prior 域构成的链表中待比较的结点
        while((q!=NULL) && (p->data>q->data)) { //查找插入结点 p 的合适的插入位置
            s=q;
            q=q->prior;
        }
        s->prior=p;
        p->prior=q;   //结点 p 插入到结点 s 和结点 q 之间
        p=p->next;
    }
}
```

6、已知线性表的元素按递增顺序排列，并以带头结点的单链表作存储结构。试编写一个删除表中所有值大于 min 且小于 max 的元素（若表中存在这样的元素）的算法。实现算法的函数原型为：void **delete**(LinkList *&head, int max, int min);（评分标准：第 1 点 4 分，第 2 点 1 分!）

答：

```
void delete(LinkList *&head, int max, int min) {
    linklist *p, *q;
    if (head!=NULL) {
        q=head;
        p=head->next;

        while((p!=NULL) && (p->data<=min)) {
            q=p;
            p=p->next;
        }

        while((p!=NULL) && (p->data<max))
            p=p->next;
```

```
            q->next=p;
        }
}
```

7、已知线性表的元素是无序的，且以带头结点的单链表作为存储结构。设计一个删除表中所有值小于 max 但大于 min 的元素的算法。实现算法的函数原型为：void ***delete***(LinkList *head, int max, int min)；（评分标准：*第 1 点 4 分，第 2 点 1 分!*）

答：

```
void delete(LinkList *&head, int max, int min) {
    LinkList *p,*q;
    q=head;
    p=head->next;
    while (p!=NULL)
        if((p->data<=min) || (p->data>=max)) {
            q=p;
            p=p->next;
        } else {
            q->next=p->next;
            free(p);
            p=q->next;
        }
}
```

8、设顺序表 L 是一个递减有序表，试写一算法，将 x 插入其后仍保持 L 的有序性。实现算法的函数原型为：int ***InsertDecreaseList***(SqList *&L, elemtype x)；（评分标准：*第 1 点 4 分，第 2 点 1 分!*）

答：（1）只要从终端结点开始往前找到第一个比 x 大(或相等)的结点数据，在这个位置插入就可以了。

（2）算法描述如下：

```
int InsertDecreaseList( SqList *&L, elemtype x ) {
    int i;
    if( (*L).len>= maxlen) {
        printf("overflow");
        return(0);
    }

    for( i=(*L).len ; i>0 && (*L).elem[ i-1 ] < x ; i--)
        (*L).elem[ i ]=(*L).elem[ i-1 ];     // 比较并移动元素

    (*L).elem[ i ] =x;
    (*L).len++;
```

```
    return(1);
}
```

9、HL 是单链表的头指针，试写出删除头结点的算法。实现算法的函数原型为：ElemType ***DeleFront***(LNode *& & HL);（评分标准：*第 1 点 4 分，第 2 点 1 分!*）

答：
```
ElemType DeleFront(LNode * & HL) {
    if (HL==NULL) {
        cerr<<"空表"<<endl;
        exit(1);
    }

    LNode* p=HL;
    HL=HL->next;
    ElemType temp=p->data;
    delete p;

    return temp;
}
```

10、统计出单链表 HL 中结点的值等于给定值 X 的结点数。实现算法的函数原型为：int ***CountX***(LNode *& HL,ElemType x);（评分标准：*第 1 点 4 分，第 2 点 1 分!*）

答：
```
int CountX(LNode* HL, ElemType x) {
    int i=0;
    LNode* p=HL;//i 为计数器
    while(p!=NULL) {
        if (P->data==x) i++;
        p=p->next;
    }//while，出循环时 i 中的值即为 x 结点个数

    return i;
}
```

11、编写算法，将一个结点类型为 Lnode 的单链表按逆序链接，即若原单链表中存储元素的次序为 a_1，……a_{n-1}，a_n，则逆序链接后变为，a_n，a_{n-1}，……a_1。实现算法的函数原型为：void ***contrary***(Lnode *& HL);（评分标准：*第 1 点 4 分，第 2 点 1 分!*）

答：
```
void contrary(Lnode * HL) {
    Lnode *P=HL;
    HL=NULL;
    While(p!=null) {
        Lnode*q=p;
        P=p->next;
        q->next=HL;
```

```
        HL=q;
    }
}
```

12、设计判断单链表中结点是否关于中心对称算法。实现算法的函数原型为：int ***lklistsymmetry***(lklist *& head)；（评分标准：*第1点4分，第2点1分!*）

答：

```
typedef struct {
    int s[100];
    int top;
} sqstack;

int lklistsymmetry(lklist *& head) {
    sqstack stack;
    stack.top = -1;

    lklist *p;
    for(p=head; p!=0; p=p->next) {
        stack.top++;
        stack.s[stack.top]=p->data;
    }

    for(p=head; p!=0; p=p->next)
        if (p->data==stack.s[stack.top])
            stack.top=stack.top-1;
        else
            return(0);

    return(1);
}
```

13、设计在单链表中删除值相同的多余结点的算法。实现算法的函数原型为：void ***delredundant***(lklist *& &head)；（评分标准：*第1点4分，第2点1分!*）

答：

```
typedef int datatype;

typedef struct node {
    datatype data;
    struct node *next;
} lklist;

void delredundant(lklist *&head) {
    lklist *p,*q,*s;
    for(p=head; p!=0; p=p->next) {
```

```
        for(q=p->next,s=q; q!=0; )
            if(q->data==p->data) {
                s->next=q->next;
                free(q);
                q=s->next;
            }else {
                s=q,q=q->next;
            }
    }
}
```

14、设单链表中有仅三类字符的数据元素(大写字母、数字和其它字符)，要求利用原单链表中结点空间设计出三个单链表的算法，使每个单链表只包含同类字符。实现算法的函数原型为：void ***split***(lklist *& head,lklist *&&ha, lklist *&&hb, lklist *&&hc);（评分标准：*第1点4分，第2点1分!*）

答：

```
typedef char datatype;

typedef struct node {
    datatype data;
    struct node *next;
} lklist;

void split(lklist *& head, lklist *& ha, lklist *& hb, lklist *& hc) {
    lklist *p;
    ha=0,hb=0,hc=0;
    for(p=head; p!=0; p=head){
        head=p->next;
        p->next=0;
        if(p->data>='A' && p->data<='Z'){
            p->next=ha;
            ha=p;
        }else if(p->data>='0' && p->data<='9'){
            p->next=hb;
            hb=p;
        }else{
            p->next=hc;
            hc=p;
        }
    }
}
```

15、设计判断单链表中元素是否是递增的算法。实现算法的函数原型为：int ***isriselk***(lklist *&

head)；（评分标准：*第 1 点 4 分，第 2 点 1 分!*）

答：

```
int isriselk(lklist *& head) {
    if( head == 0 || head->next == 0 )
        return(1);
    else
        for(q=head, p=head->next;   p!=0;   q=p, p=p->next)
            if( q->data>p->data )
                return(0);

    return(1);
}
```

得分

七、分析题（X 题，每题 X 分，共 X 分）

评分标准：每题回答完全正确得 5 分，其余按得分点给分！

1、在如下数组 A 中链接存储了一个线性表，表头指针为 A[0].next，试写出该线性表。（评分标准：*第 1 点 4 分，第 2 点 1 分!*）

A	0	1	2	3	4	5	6	7	8
data		60	50	78	90	34		40	
next	3	5	7	2	0	4		1	

答：线性表为：（78，50，40，60，34，90）

得分

八、简答题（X 题，每题 X 分，共 X 分）

评分标准：每题回答完全正确得 5 分，其余按得分点给分！

1、描述以下三个概念的区别：头指针，头结点，表头结点。（评分标准：*第 1 点 4 分，第 2 点 1 分!*）

答：**头指针**：是指向链表中第一个结点（即表头结点）的指针；

头结点：在表头结点之前附设的结点称为头结点；

表头结点：为链表中存储线性表中第一个数据元素的结点。

若链表中附设头结点，则不管线性表是否为空表，头指针均不为空，否则表示空表的链表的头指针为空。

2、线性表的两种存储结构各有哪些优缺点？（评分标准：*第 1 点 4 分，第 2 点 1 分!*）

答：线性表具有两种存储结构即顺序存储结构和链接存储结构。

线性表的顺序存储结构可以直接存取数据元素，方便灵活、效率高，但插入、删除操作时将会引起元素的大量移动，因而降低效率：

而在链接存储结构中内存采用动态分配，利用率高，但需增设指示结点之间关系的指针域，存取数据元素不如顺序存储方便，但结点的插入、删除操作较简单。

3、对于线性表的两种存储结构，如果有 n 个线性表同时并存，而且在处理过程中各表的长度会动态发生变化，线性表的总数也会自动改变，在此情况下，应选用哪一种存储结构？为什么？（评分标准：*第 1 点 4 分，第 2 点 1 分!*）

答：应选用链接存储结构，因为链式存储结构是用一组任意的存储单元依次存储线性表中的各元素，这里存储单元可以是连续的，也可以是不连续的：这种存储结构对于元素的删除或插入运算是不需要移动元素的，只需修改指针即可，所以很容易实现表的容量的扩充。

4、对于线性表的两种存储结构，若线性表的总数基本稳定，且很少进行插入和删除操作，但要求以最快的速度存取线性表中的元素，应选用何种存储结构？试说明理由。（评分标准：*第 1 点 4 分，第 2 点 1 分!*）

答：应选用顺序存储结构，因为每个数据元素的存储位置和线性表的起始位置相差一个和数据元素在线性表中的序号成正比的常数。因此，只要确定了其起始位置，线性表中的任一个数据元素都可随机存取，因此，线性表的顺序存储结构是一种随机存取的存储结构，而链表则是一种顺序存取的存储结构

5、在单循环链表中设置尾指针比设置头指针好吗？为什么？（评分标准：*第 1 点 4 分，第 2 点 1 分!*）

答：设尾指针比设头指针好。尾指针是指向终端结点的指针，用它来表示单循环链表可以使得查找链表的开始结点和终端结点都很方便，设一带头结点的单循环链表，其尾指针为 rear，则开始结点和终端结点的位置分别是 rear->next->next 和 rear，查找时间都是 O(1)。若用头指针来表示该链表，则查找终端结点的时间为 O(n)。

6、下述算法的功能是什么？（评分标准：*第 1 点 4 分，第 2 点 1 分!*）

```
LinkList * Demo(LinkList *& L) {// L 是无头结点的单链表
    LinkList *q,*p;
    if(L&&L->next) {
        q=L;
        L=L->next;
        p=L;

        while (p->next)
            p=p->next;

        p->next=q;
        q->next=NULL;
    }
    return (L);
}
```

答：该算法的功能是：将开始结点摘下链接到终端结点之后成为新的终端结点，而原来的第二个结点成为新的开始结点，返回新链表的头指针。