第 2 章　线性表

1．选择题

（1）顺序表中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

答案：B

解释：顺序表中的数据连续存储，所以第 5 个元素的地址为：100+2*4=108。

（2）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（　　）。

A．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）

B．在第 i 个结点后插入一个新结点（1≤i≤n）

C．删除第 i 个结点（1≤i≤n）

D．将 n 个结点从小到大排序

答案：A

解释：在顺序表中插入一个结点的时间复杂度都是 $O(n^2)$，排序的时间复杂度为 $O(n^2)$或 $O(n\log_2 n)$。顺序表是一种随机存取结构，访问第 i 个结点和求第 i 个结点的直接前驱都可以直接通过数组的下标直接定位，时间复杂度是 O(1)。

（3） 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__的元素个数为（　　）。

A．8　　B．63.5　　C．63　　D．7

答案：B

解释：平均要移动的元素个数为：n/2。

（4）链接存储的存储结构所占存储空间（　　）。

A．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

B．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

答案：A

（5）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

答案：D

（6）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

答案：B

解释：链表最大的优点在于插入和删除时不需要移动数据，直接修改指针即可。

（7）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

答案：C

解释：存储密度是指一个结点数据本身所占的存储空间和整个结点所占的存储空间之比，假设单链表一个结点本身所占的空间为 D，指针域所占的空间为 N，则存储密度为：D/(D+N)，一定小于 1。

（8）将两个各有 n 个元素的有序表归并成一个有序表，其最少的比较次数是（　　）。

A．n　　B．2n-1　　C．2n　　D．n-1

答案：A

解释：当第一个有序表中所有的元素都小于（或大于）第二个表中的元素，只需要用第二个表中的第一个元素依次与第一个表的元素比较，总计比较 n 次。

（9）在一个长度为 n 的顺序表中，在第 i 个元素（1≤i≤n+1）之前插入一个新元素时须向后移动（　　）个元素。

A．n-i　　B．n-i+1　　C．n-i-1　　D．I

答案：B

(10) 线性表 L=(a_1，a_2,……a_n)，下列说法正确的是（　　）。

A．每个元素都有一个直接前驱和一个直接后继

B．线性表中至少有一个元素

C．表中诸元素的排列必须是由小到大或由大到小

D．除第一个和最后一个元素外，其余每个元素都有一个且仅有一个直接前驱和直接后继。

答案：D

(11) 创建一个包括 *n* 个结点的有序单链表的时间复杂度是（　　）。

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

答案：C

解释：单链表创建的时间复杂度是 O(n)，而要建立一个有序的单链表，则每生成一个新结点时需要和已有的结点进行比较，确定合适的插入位置，所以时间复杂度是 O(n2)。

(12) 以下说法错误的是（　　）。

A．求表长、定位这两种运算在采用顺序存储结构时实现的效率不比采用链式存储结构时实现的效率低

B．顺序存储的线性表可以随机存取

C．由于顺序存储要求连续的存储区域，所以在存储管理上不够灵活

D．线性表的链式存储结构优于顺序存储结构

答案：D

解释：链式存储结构和顺序存储结构各有优缺点，有不同的适用场合。

(13) 在单链表中，要将 s 所指结点插入到 p 所指结点之后，其语句应为（　　）。

A．s->next=p+1; p->next=s;

B．(*p).next=s; (*s).next=(*p).next;

C．s->next=p->next; p->next=s->next;

D．s->next=p->next; p->next=s;

答案：D

(14) 在双向链表存储结构中，删除 p 所指的结点时须修改指针（　　）。

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

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

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

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

答案：A

(15) 在双向循环链表中，在 p 指针所指的结点后插入 q 所指向的新结点，其修改指针的操作是（　　）。

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

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

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

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

答案：C

2．算法设计题

（1）将两个递增的有序链表合并为一个递增的有序链表。要求结果链表仍使用原来两个链表的存储空间，不另外占用其它的存储空间。表中不允许有重复的数据。

[题目分析]

合并后的新表使用头指针 Lc 指向，pa 和 pb 分别是链表 La 和 Lb 的工作指针,初始化为相应链表的第一个结点，从第一个结点开始进行比较，当两个链表 La 和 Lb 均为到达表尾结点时，依次摘取其中较小者重新链接在 Lc 表的最后。如果两个表中的元素相等，只摘取 La 表中的元素，删除 Lb 表中的元素，这样确保合并后表中无重复的元素。当一个表到达表尾结点，为空时，将非空表的剩余元素直接链接在 Lc 表的最后。

[算法描述]

```
void MergeList(LinkList &La,LinkList &Lb,LinkList &Lc)
{//合并链表 La 和 Lb，合并后的新表使用头指针 Lc 指向
  pa=La->next;  pb=Lb->next;
   //pa 和 pb 分别是链表 La 和 Lb 的工作指针,初始化为相应链表的第一个结点
   Lc=pc=La;  //用 La 的头结点作为 Lc 的头结点
   while(pa && pb)
{if(pa->data<pb->data){pc->next=pa;pc=pa;pa=pa->next;}
     //取较小者 La 中的元素，将 pa 链接在 pc 的后面，pa 指针后移
     else if(pa->data>pb->data) {pc->next=pb; pc=pb; pb=pb->next;}
      //取较小者 Lb 中的元素，将 pb 链接在 pc 的后面，pb 指针后移
     else //相等时取 La 中的元素，删除 Lb 中的元素
{pc->next=pa;pc=pa;pa=pa->next;
      q=pb->next;delete pb ;pb =q;
}
     }
 pc->next=pa?pa:pb;     //插入剩余段
     delete Lb;             //释放 Lb 的头结点
}
```

（2）将两个非递减的有序链表合并为一个非递增的有序链表。要求结果链表仍使用原来两个链表的存储空间，不另外占用其它的存储空间。表中允许有重复的数据。

[题目分析]

合并后的新表使用头指针 Lc 指向，pa 和 pb 分别是链表 La 和 Lb 的工作指针,初始化为相应链表的第一个结点，从第一个结点开始进行比较，当两个链表 La 和 Lb 均为到达表尾结点时，依次摘取其中较小者重新链接在 Lc 表的表头结点之后，如果两个表中的元素相等，只摘取 La 表中的元素，保留 Lb 表中的元素。当一个表到达表尾结点，为空时，将非空表的剩余元素依次摘取，链接在 Lc 表的表头结点之后。

[算法描述]

```
void MergeList(LinkList& La, LinkList& Lb, LinkList& Lc, )
{//合并链表 La 和 Lb，合并后的新表使用头指针 Lc 指向
  pa=La->next;  pb=Lb->next;
//pa 和 pb 分别是链表 La 和 Lb 的工作指针,初始化为相应链表的第一个结点
  Lc=pc=La; //用 La 的头结点作为 Lc 的头结点
  Lc->next=NULL;
  while(pa||pb )
{//只要存在一个非空表，用 q 指向待摘取的元素
    if(!pa)  {q=pb;  pb=pb->next;}
//La 表为空，用 q 指向 pb，pb 指针后移
    else if(!pb)  {q=pa;  pa=pa->next;}
//Lb 表为空，用 q 指向 pa，pa 指针后移
    else if(pa->data<=pb->data)  {q=pa;  pa=pa->next;}
//取较小者（包括相等）La 中的元素，用 q 指向 pa，pa 指针后移
    else {q=pb;  pb=pb->next;}
//取较小者 Lb 中的元素，用 q 指向 pb，pb 指针后移
     q->next = Lc->next;  Lc->next = q;
//将 q 指向的结点插在 Lc 表的表头结点之后
    }
    delete Lb;              //释放 Lb 的头结点
}
```

（3）已知两个链表 A 和 B 分别表示两个集合，其元素递增排列。请设计算法求出 A 与 B 的交集，并存放于 A 链表中。

[题目分析]

只有同时出现在两集合中的元素才出现在结果表中,合并后的新表使用头指针 Lc 指向。pa 和 pb 分别是链表La和Lb的工作指针,初始化为相应链表的第一个结点,从第一个结点开始进行比较，当两个链表 La 和 Lb 均为到达表尾结点时，如果两个表中相等的元素时，摘取 La 表中的元素，删除 Lb 表中的元素；如果其中一个表中的元素较小时，删除此表中较小的元素，此表的工作指针后移。当链表 La 和 Lb 有一个到达表尾结点，为空时，依次删除另一个非空表中的所有元素。

[算法描述]

```
void Mix(LinkList& La, LinkList& Lb, LinkList& Lc)
{  pa=La->next;pb=Lb->next;
pa 和 pb 分别是链表 La 和 Lb 的工作指针,初始化为相应链表的第一个结点
Lc=pc=La; //用 La 的头结点作为 Lc 的头结点
while(pa&&pb)
{ if(pa->data==pb->data)//交集并入结果表中。
   { pc->next=pa;pc=pa;pa=pa->next;
     u=pb;pb=pb->next; delete u;}
  else if(pa->data<pb->data) {u=pa;pa=pa->next; delete u;}
else {u=pb; pb=pb->next; delete u;}
}
while(pa) {u=pa; pa=pa->next; delete u;}// 释放结点空间
while(pb) {u=pb; pb=pb->next; delete u;}//释放结点空间
```

```
pc->next=null;//置链表尾标记。
delete Lb;  //释放 Lb 的头结点
   }
```

（4）已知两个链表 A 和 B 分别表示两个集合，其元素递增排列。请设计算法求出两个集合 A 和 B 的差集（即仅由在 A 中出现而不在 B 中出现的元素所构成的集合），并以同样的形式存储，同时返回该集合的元素个数。

[题目分析]

求两个集合 A 和 B 的差集是指在 A 中删除 A 和 B 中共有的元素，即删除链表中的相应结点,所以要保存待删除结点的前驱，使用指针 pre 指向前驱结点。pa 和 pb 分别是链表 La 和 Lb 的工作指针,初始化为相应链表的第一个结点，从第一个结点开始进行比较，当两个链表 La 和 Lb 均为到达表尾结点时，如果 La 表中的元素小于 Lb 表中的元素，pre 置为 La 表的工作指针 pa 删除 Lb 表中的元素；如果其中一个表中的元素较小时，删除此表中较小的元素，此表的工作指针后移。当链表 La 和 Lb 有一个为空时，依次删除另一个非空表中的所有元素。

[算法描述]

```
void Difference（LinkList& La, LinkList& Lb,int *n）
{//差集的结果存储于单链表 La 中，*n 是结果集合中元素个数，调用时为 0
pa=La->next; pb=Lb->next;
//pa 和 pb 分别是链表 La 和 Lb 的工作指针,初始化为相应链表的第一个结点
  pre=La;            //pre 为 La 中 pa 所指结点的前驱结点的指针
  while（pa&&pb）
{if（pa->data<q->data）{pre=pa;pa=pa->next;*n++;}
//  A 链表中当前结点指针后移
else if（pa->data>q->data）q=q->next;      //B 链表中当前结点指针后移
    else {pre->next=pa->next;       //处理 A，B 中元素值相同的结点，应删除
          u=pa; pa=pa->next; delete u;}    //删除结点
}
}
```

（5）设计算法将一个带头结点的单链表 A 分解为两个具有相同结构的链表 B、C，其中 B 表的结点为 A 表中值小于零的结点，而 C 表的结点为 A 表中值大于零的结点（链表 A 中的元素为非零整数，要求 B、C 表利用 A 表的结点）。

[题目分析]

B 表的头结点使用原来 A 表的头结点，为 C 表新申请一个头结点。从 A 表的第一个结点开始，依次取其每个结点 p，判断结点 p 的值是否小于 0，利用前插法，将小于 0 的结点插入 B 表,大于等于 0 的结点插入 C 表。

[算法描述]

```
void DisCompose(LinkedList A)
{     B=A;
B->next= NULL;     //B 表初始化
     C=new LNode;//为 C 申请结点空间
     C->next=NULL;      //C 初始化为空表
     p=A->next;         //p 为工作指针
     while(p!= NULL)
     { r=p->next;       //暂存 p 的后继
       if(p->data<0)
```

```
        {p->next=B->next; B->next=p; }//将小于 0 的结点链入 B 表,前插法
      else {p->next=C->next; C->next=p; }//将大于等于 0 的结点链入 C 表,前插法
      p=r;//p 指向新的待处理结点。
    }
}
```

（6）设计一个算法，通过一趟遍历在单链表中确定值最大的结点。

[题目分析]

假定第一个结点中数据具有最大值，依次与下一个元素比较，若其小于下一个元素，则设其下一个元素为最大值，反复进行比较，直到遍历完该链表。

[算法描述]

```
ElemType Max (LinkList L ){
    if(L->next==NULL) return NULL;
    pmax=L->next; //假定第一个结点中数据具有最大值
    p=L->next->next;
    while(p != NULL ){//如果下一个结点存在
        if(p->data > pmax->data) pmax=p;//如果 p 的值大于 pmax 的值，则重新赋值
        p=p->next;//遍历链表
    }
    return pmax->data;
```

（7）设计一个算法，通过遍历一趟，将链表中所有结点的链接方向逆转，仍利用原表的存储空间。

[题目分析]

从首元结点开始，逐个地把链表 L 的当前结点 p 插入新的链表头部。

[算法描述]

```
void  inverse(LinkList &L)
{// 逆置带头结点的单链表 L
    p=L->next;  L->next=NULL;
    while ( p) {
        q=p->next;     // q 指向*p 的后继
        p->next=L->next;
        L->next=p;        // *p 插入在头结点之后
        p = q;
    }
}
```

（8）设计一个算法，删除递增有序链表中值大于 mink 且小于 maxk 的所有元素（mink 和 maxk 是给定的两个参数，其值可以和表中的元素相同，也可以不同 ）。

[题目分析]

分别查找第一个值>mink 的结点和第一个值 ≥maxk 的结点，再修改指针，删除值大于 mink 且小于 maxk 的所有元素。

[算法描述]

```
void delete(LinkList &L, int mink, int maxk) {
   p=L->next; //首元结点
   while (p && p->data<=mink)
```

```
      { pre=p;  p=p->next; } //查找第一个值>mink 的结点
   if (p)
{while (p && p->data<maxk)  p=p->next;
                    // 查找第一个值 ≥maxk 的结点
      q=pre->next;   pre->next=p;  // 修改指针
      while (q!=p)
         { s=q->next;  delete q;  q=s; } // 释放结点空间
   }//if
}
```

（9）已知 p 指向双向循环链表中的一个结点，其结点结构为 data、prior、next 三个域，写出算法 change(p),交换 p 所指向的结点和它的前缀结点的顺序。

[题目分析]

知道双向循环链表中的一个结点，与前驱交换涉及到四个结点（p 结点，前驱结点，前驱的前驱结点，后继结点）六条链。

[算法描述]

```
void  Exchange（LinkedList p）
//p 是双向循环链表中的一个结点，本算法将 p 所指结点与其前驱结点交换。
{q=p->llink;
 q->llink->rlink=p;     //p 的前驱的前驱之后继为 p
 p->llink=q->llink;     //p 的前驱指向其前驱的前驱。
 q->rlink=p->rlink;     //p 的前驱的后继为 p 的后继。
 q->llink=p;            //p 与其前驱交换
 p->rlink->llink=q;     //p 的后继的前驱指向原 p 的前驱
 p->rlink=q;            //p 的后继指向其原来的前驱
}//算法 exchange 结束。
```

（10）已知长度为 n 的线性表 A 采用顺序存储结构，请写一时间复杂度为 O(n)、空间复杂度为 O(1)的算法，该算法删除线性表中所有值为 item 的数据元素。

[题目分析]

在顺序存储的线性表上删除元素，通常要涉及到一系列元素的移动（删第 i 个元素，第 i+1 至第 n 个元素要依次前移）。本题要求删除线性表中所有值为 item 的数据元素，并未要求元素间的相对位置不变。因此可以考虑设头尾两个指针（i=1，j=n），从两端向中间移动，凡遇到值 item 的数据元素时，直接将右端元素左移至值为 item 的数据元素位置。

[算法描述]

```
void  Delete（ElemType A[ ]，int  n）
//A 是有 n 个元素的一维数组，本算法删除 A 中所有值为 item 的元素。
{i=1；j=n；//设置数组低、高端指针（下标）。
 while（i<j）
   {while（i<j && A[i]!=item）i++;          //若值不为 item，左移指针。
    if（i<j）while（i<j && A[j]==item）j--；//若右端元素为 item，指针左移
    if（i<j）A[i++]=A[j--]；}
```