期末题大题：线性表&&栈和队列

Object Group

栈和队列

出栈顺序类

1、设有一个输入序列 abcd，元素经过一个栈到达输出序列，并且元素一旦离开输入序列就不能再回到输入序列，试问经过这个栈后可以得到多少种输出序列？ (4 分)
解：元素可以入栈，然后出栈到达输出序列，也可以入栈后停留，然后再出栈到达输出序列。所有可能的输出序列为：

以 a 为开头： abcd，abdc，acdb，acbd，adcb
以 b 为开头： bacd，badc，bcad，bcda，bdca
以 c 为开头： cbad，cbda，cdba
以 d 为开头： dcba

2.（6 分）设有编号为 1，2，3，4 的四辆列车，顺序进入一个栈式结构的车站，具体写出这四辆列车开出车站的所有可能的顺序。

答：至少有 14 种。 （酌情给分）

① 全进之后再出情况，只有 1 种：4，3，2，1

② 进 3 个之后再出的情况，有 3 种，3,4,2,1 3,2,4,1 3,2,1,4

③ 进 2 个之后再出的情况，有 5 种，2,4,3,1 2,3,4,1 2,1, 3,4 2,1,4,3 2,1,3,4

④ 进 1 个之后再出的情况，有 5 种，1,4,3,2 1,3,2,4 1,3,4,2 1, 2,3,4 1,2,4,3

栈的基本操作

2．(10 分)一个双向栈 S 是在同一向量空间内实现的两个栈(顺序栈)，它们的栈底分别设在向量空间的两端。如下图所示。 试为此双向栈设计初始化InitStack (S) 、入栈 Push(S , i , x) 和出栈 Pop(S , i)等三个子操作算法， 其中i 为 0 或 1，用以表示栈号。

双向栈 S


```
//双向栈数据类型定义
#define STACK_SIZE 100; Typedef struct
{
    SElemType * base_one, * base_two;//栈底指针
    ElemType  * top_one, * top_two;//栈顶指针          int stacksize;
} SqStack;

Status InitStack ( SqStack &S)
{ //初始化双向栈
    //第一个栈底和栈顶指向数组起点
```

```
    S.base_one=S.top_one=( SElemType *)malloc(STACK_SIZE * sizeof(SElemType));
    // 第二个栈底和栈顶指向数组终点
    S.base_two=S.top_two=S.base_one +STACK_SIZE-1;
    S.stacksize= STACK_SIZE;
    return OK;
}//InitStack
Status Push ( SqStack &S, int i, SElemType e)
{ //入栈操作，i=0 时将 e 存入前栈，i=1 时将 e 存入后栈
    if( S.top_two < S.top_one) return OVERFLOW;//栈满，不考虑追加空间
    if( i = = 0 )
        *S.top_one++ = e;
    else
        *S.top_two-- = e;
    return OK;
}//Push

SElemType Pop ( SqStack &S, int i)
{   //出栈操作，i=0 出前栈，i=1 出后栈
    if( i==0 )
    {
        if ( S.top_one==S.base_one) return ERROR;
        S.top_one--; e=*S.top_one;
    }
    else
    {
        if ( S.top_two==S.base_two) return ERROR;
        S.top_two++; e=*S.top_two;
    }
    return e;
}//Pop
```

2.（10 分）假设下面程序里的栈 S 存放如下表左列的数据，经过下面程序处理后，写出栈内的内容（写在表右列里），并说明该程序的功能。

1000	
800	
600	
400	
200	
100	

```
void Demo(struct Stack *S)
{
   int i;
   int arr[64];
   n=0;
   while（! EMPTYS(S)） arr[n++]=POPS(S);
   for(i=0;i<n;i++) PUSHS(S, arr[i]);
}
```

2. (10 分)

1000	100
800	200
600	400
400	600
200	800
100	1000

（5 分）

程序功能：先将栈 S 中的内容全部出栈放入 arr 数组中；然后再将 arr 数组内容顺序入栈至 S。（5 分）

队列的基本操作

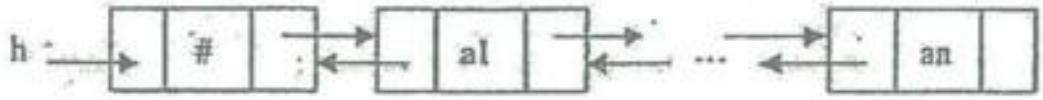
1、以下运算实现在循环队上的入队列，请在_____处用适当句子予以填充。

```
Int EnCycQueue(CycquereTp *sq,DataType x)
    {  if((sq->rear+1)%maxsize== __ sq->front __)
            {error("队满");return(0);
       else{ ___ sq->rear=(sq->rear+1)%maxsize_____;
             __ sq->data[sq->rear]=x_____;
            return(1);
    }
```

1.（10 分）链队列的示意图如下所示：

（1）用 C 语言描述其结构类型。

（2）写出在头结点 h 后插入一个结点 i 的算法。



1.（10 分）

（1）

```
typedef struct dnode
{ datatype data;
  struct dnode  *prior, *next;
}dlinklist;
dlinklist  *h;
```

（4 分）

（2）

```
i-> prior = h;
i->next = h->next;
h->next-> prior = i;
h->next = i;
```

（2 分）

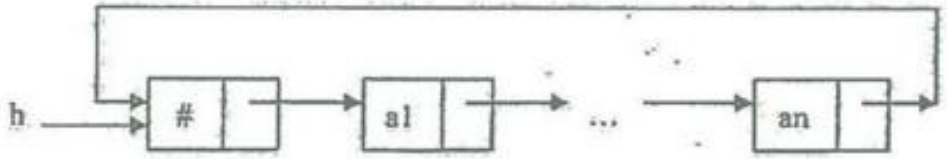
（2 分）

（2 分）

1.（10 分）链队列的示意图如下所示：

（1）用 C 语言描述其结构类型。

（2）写出在头结点 h 后插入一个结点 i 的算法。



1．（10 分）

解：

（1）

```
typedef struct node
{
    datatype data;
    struct node *next;
}linklist;
linklist   *h;                                  （4 分）
```

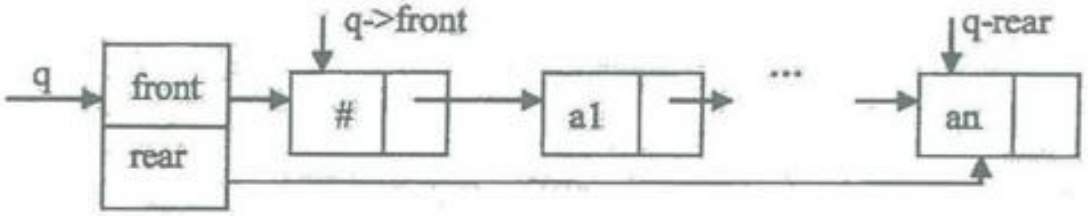
（2）

```
i->next = h->next;                              （3 分）
h->next = i;                                    （3 分）
```

1.（10 分）链队列的示意图如下所示。

（1）用 C 语言描述其结构类型。

（2）写出删除开始结点（即头结点之后的第一个结点）的算法。



1．(10 分)

(1) typedef struct node

```
    {   datatype data;
        struct node *next;
    }linklist;
    typedef struct
    {   linklist *front,*rear;
    }linkqueue *q;
```

（2 分）

（2 分）

(2)int DEQUEUEQL(linkqueue *q)

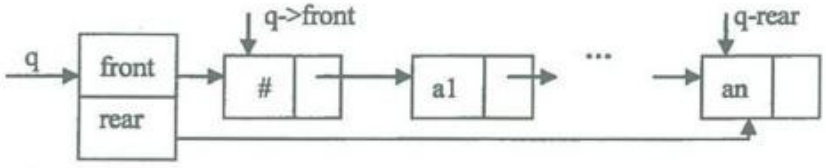
```
  {  linklist *s;
     if(EMPTY(q))
     {  printf("队列空！\n");
        return(0);
     }
     else
     {  s=q->front->next;          /* 指向被删除头结点 */
        if(s->next==NULL)          /* 当前链队列长度等于 1 */
        {  q->front->next=NULL;
           q->rear=q->front;
        }
        else  q->front->next=s->next;
        free(s);
        return(1);
     }
  }
```

（2 分）

（2 分）

（2 分）

1.（10 分）链队列的示意图如下所示。（1）用 C 语言描述其结构类型；（2）写出队列的删除算法。



1. (10 分)

(1) typedef struct node

```
{   datatype data;
    struct node *next;
}linklist;
typedef struct
{   linklist *front,*rear;
}linkqueue *q;
```

(4 分)

(2)datatype DEQUEUEQL(linkqueue *q)

```
{   datatype temp;
    if(EMPTY(q))
    {   printf("队列空！ \n");
        return(-1);
    }
    else
    {   s=q->front->next;          /* 指向被删除头结点 */
        if(s->next==NULL)          /* 当前链队列长度等于 1 */
        {   q->front->next=NULL;
            q->rear=q->front;
        }
        else   q->front->next=s->next;     /* 修改头结点指针 */
        temp=s->data;
        return(temp);                      /* 返回被删除结点的值 */
    }
}
```

(6 分)

注意不需要损失一个空间的循环队列

2. (10 分) 假设以数组 sequ[m]存放循环队列的元素，同时设变量 rear 和 quelen 分别指示循环队列中队尾元素的位置和内含元素的个数。写出相应的入队列和出队列的算法(在出队的算法中要返回队头元素)。

2. (10 分)

解：

```
#define M 128
typedef  struct queue
{  datatype      sequ[M];
   int quelen;
   int rear;
```

} sequeue; （2 分）

```
int ENQUEUE (sequeue *sq , datatype x)
{
        if (sq->quelen==M) return (0);
        sq->rear=(sq->rear+1)%M;
        seque[sq->rear] =x ;
        sq->quelen++;
        return(1);
}                                   （4 分）
int DEQUEUE (sequeue *sq)
{       int  p;
        if (sq->quelen==0) return (-1);
        p=(sq->rear - sq->quelen +1+M) %M;
        sq->quelen- -;
        return(p);
 }                                  （4 分）
```

2. （10 分） 如果用一个循环数组 q[0..m-1]表示队列时，该队列只有一个队列头指针 front，不设队列尾指针 rear，而改置计数器 count 用以记录队列中结点的个数。

（1）写出其数据结构。

（2）编写实现队列的三个基本运算：判空、入队、出队。

（3）队列中能容纳元素的最多个数是多少？

2. (10 分)

(1)typedef struct

```
{elemtp q[m];
 int front,count;  //front 是队首指针，count 是队列中元素个数。
}cqnode;           //定义类型标识符。
```

(3 分)

(2)判空：int Empty(cqnode cq) //cq 是 cqnode 类型的变量

```
{if(cq.count==0) return(1); else return(0);  //空队列}
```

(2 分)

入队: int EnQueue(cqnode cq，elemtp x)

```
{if(count==m){printf("队满\n"); exit(0); }
 count++;
 cq.q[(cq.front+count)%m]=x;     //x 入队
 return(1);                      //队列中元素个数增加 1,入队成功。
 }
```

(2 分)

出队: int DelQueue(cqnode cq)

```
{if (count==0){printf("队空\n"); return(0);}
 printf("出队元素", cq.q[cq.front]);
 cq.front=(cq.front+1)%m; //计算新的队头指针。
 x=cq.q[cq.front];
 return(x)
}
```

(2 分)

(3) 队列中能容纳的元素的个数为 m。 (1 分)

Object Group

线性表

合并与分解

1.（8 分）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 的元素类型为整型，要求 B、C 表利用 A 表的结点）。

```
void decompose(Linklist La,Linklist &Lb,Linklist &Lc)
{
Linknode *p;
Lc=(Linknode *)malloc(sizeof(Linknode));
Lc->next=NULL;
p=La->next;
Lb=La;
Lb->next=NULL;
while(p)
{
    La=p->next;
```

```
        if(p->data>0)
        {
            p->next=Lc->next;
            Lc->next=p;
        }
        else
        {
            p->next=Lb->next;
            Lb->next=p;
        }
        p=La;
    }
}
```

1、设线性表 A = (a_1, a_2, …,a_m)，B= (b_1, b_2, …,b_n)，试写一个按下列规则将线性表 A、B 合并为线性表 C 的算法，使得

C= (a_1, b_1, …, a_m ,b_m ,b_{m+1}, …,b_n) 当 m≤n 时；

C= (a_1, b_1, …, a_n ,b_n ,a_{n+1}, …,a_m) 当 m>n 时。

线性表 A、B 和 C 均以带头结点的单链表作为存储结构，且 C 表利用 A 表和 B 表中的结点空间构成，C 表的头结点另辟空间。请将算法补充完整。

```
typedef  struct  node
{   datatype  data;  //数据域
    struct  node *next;  //指针域
}LinkList;
LinkList* merge(LinkList* A, LinkList* B);
```

```
//A、B 分别指向 A 表、B 表，返回 C 表的指针
{   LinkList*p, *q;
    LinkList*C=(LinkList*)malloc(sizeof(LinkList));
    C->next=NULL; q=C;
    while(A->next!=NULL&&B->next!=NULL)
    {   p=A->next;
        ①_____
        q->next=p;   q=q->next;
        p=B->next;
        ②_____
        q->next=p;   q=q->next;
    }
    if( A->next!=NULL )  ③_____
    if( B->next!=NULL )  ④_____
    return  C;
}
```

1. ①A->next=p->next;
②B->next=p->next;
③q->next=A->next
④q->next=B->next;

2、以下将 $a_h, \cdots, a_m$ 和 $a_{m+1}, \cdots, a_n$ 两个有序序列（下标范围分别为 h——m，m+1——n，它们相应的关键字值满足 $K_h <= \cdots <= K_m$, $K_{m+1} <= \cdots <= K_n$）合并成一个有序序列 $R_h, \cdots, R_n$（使其关键字值满足 $K_h <= \cdots <= K_n$）。请分析算法，并在_____上填充适当的语句。

```
void merge(list a,list R,int h,int m,int n)
  {i=h;k=h;j=m+1;
   while((i<=m)&&(j<=n))
     {if(a[i].key<=a[j].key){R[k]=_ a[i]_;_ i++_;}
      else{R[k]=_ a[j]_;__ j++__;}
      k++;
      }
   while(i<=___m___){R[k]=a[i];i++;k++;}
   while(j<=___n___){R[k]=a[j];j++;k++;}
   }
```

线性表定位操作，值定位

1. 设计一个实现下述要求的 Locate 运算的算法：有一个双向链表，每个结点 node 有四个域，prior、next、data 和 freq。初始化时，freq 域的值都为 0，每当在链表中进行一次 Locate (s，x)运算时,令元素值为 x 的节点中 freq 域的值增加 1，并使此链表中结点保持按访问频度递减的顺序排列，以便使频繁访问的结点总是靠近表头。 (8 分)

```
DuLinkList  Locate(DuLinkList  s, ElemType  x)
    DuLinkList  p,q;
    p=s->next;
```

```
    while(p!=s &&p->data!=x)
        p=p->next;
    if(p==s)
        return NULL;
    p->freq++;
    q=p->prior;          /*向前查找*/
    while(q!=s && q->freq<p->freq)
        q=q->prior;
    if (q!=p->prior)
    {
            p->prior->next=p->next;
            p->next->prior=p->prior;
            p->next=q->next;
            q->next->prior=p;
            q->next=p;
            p->prior=q;
     }
     return p;
}
```

1.（10 分）统计不带头结点的单链表 HL 中结点的值等于给定值 X 的结点数。 该函数名称如下：int CountX(LNode* HL,ElemType x)

解：视具体情况酌情给分。

```
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;
}//CountX
```

链表排序

3. 对单链表中元素按插入方法排序的 C 语言描述算法如下，其中 L 为链表头结点指针。请填充算法中标出的空白处，完成其功能。(5 分)

```
typedef struct node
    {   int data;
        struct node *next;
    }linknode,*link;
void Insertsort(link L)
  { link p,q,r,u;
    p=L->next;   (1) p=p->next
    while((2) p!=null)
      { r=L;  q=L->next;
        while((3) q!=null && q->data<=p->data) {r=q; q=q->next;}
        u=p->next; (4) r->next=p; (5) p->next=q p=u;
      }
```

1.(10 分)假设以单链表表示线性表,单链表的类型定义如下:

```
typedef struct node {
    DataType data;
    struct node *next;
} LinkNode, *LinkList;
```

编写算法,将一个头指针为 head 且不带头结点的单链表改造为一个含头结点且头指针仍为 head 的单向循环链表，并分析算法的时间复杂度。

(1)

```
void f(LinkList &head){
    LinkNode *p=new node();
    if(head==NULL){
        head=p;
        p->next=p;
    }
    else{
        p->next=head;
        LinkList t=head;
        while(t->next!=NULL)t=t->next;
        t->next=p;
        head=p;
    }
}
```

算法时间复杂度为 O(n)

4. 已知一个带头结点的单链表，其头指针为 H，链表中数据元素 Key 为整数类型，试设计算法，将此单链表中的元素按 Key 值递增的顺序进行就地排序。（10 分）

单链表结点类型为：

```
typedef struct LNode
    { int Key; //数据域
    struct LNode   *next;   //指针域
} LNode,   *LinkList;
```

```
void Sort(LinkList &L){
//本算法实现将单链表 L 的结点重排，使其递增有序
    LNode *p=L->next,*pre;
    LNode *r=p->next;              //r 保持*p 后继结点指针，以保证不断链
    p->next=NULL;                  //构造只含一个数据结点的有序表
    p=r;
    while(p!=NULL){
        r=p->next;                 //保存*p 的后继结点指针
        pre=L;
        while(pre->next!=NULL&&pre->next->data<p->data)
            pre=pre->next;         //在有序表中查找插入*p 的前驱结点*pre
        p->next=pre->next;         //将*p 插入到*pre 之后
        pre->next=p;
        p=r;                       //扫描原单链表中剩下的结点
    }
}
```

插入删除逆置

1. 已知一带头结点的非空单链表， L 为该链表的头指针，指针 P 所指的结点既不是第一个结点，也不是最后一个结点，试写出删除 P 结点的语句序列。（6 分）

单链表结点类型为：

```
typedef   struct   LNode {
                ElemType   data; //数据域
                struct   LNode   *next;   //指针域
            } LNode,   *LinkList;
```

1.

```
LinkList q=L->next;
While(q->next!=p&&q)
{
        q=q->next;
}
q->next=p->next;
```

1. 已知长度为 n 的线性表 A=(a_1, a_2, …, a_{n-1}, a_n) 采用顺序存储结构。编写一算法，将线性表原地转换为 A'=(a_n, a_{n-1}, …, a_2, a_1)，要求转换过程中用尽可能少的辅助空间。(5 分)

算法思路：只需从线性表的第一个数据元素开始，将第 i 个数据元素与第 n-i+1 个元素交换位置即可(两元素交换位置可以通过三条赋值语句实现)。在这个过程中，i 的变化范围是 1 至 n/2 下取整。只用到了一个辅助空间 temp。

算法：

```
void REVERSE(int A[], int n)          (1 分)
{
  for(int i=0; i<n/2; i++){           (2 分)
    int temp=A[i];
    A[i]=A[n-i-1];                    (2 分)
    A[n-i-1]=temp;
  }
}
```

1.（10 分）试写一算法，将带头结点的单链表逆置，即将原始的单链表（a_1, a_2, . . . a_{n-1}, a_n）逆序变为（a_n, a_{n-1}, . . . , a_2, a_1）。

单链表结点类型定义为：

```
typedef  struct  LNode {
         ElemType  data; //数据域
         struct  LNode  *next;  //指针域
} LNode,  *LinkList;
```

1.（10 分）写出单链表的建表算法。

1. (10 分)

单链表数据结构：

```
typedef struct node
      {   datatype data;
          struct node *next;
      }linklist;
```

（3 分）

单链表的建表算法为头插法、尾插法或带头结点的尾插法均可。

(1) 头插法：

```
linklist *CREATLISTF( )
{ char ch;      /* 逐个输入字符，以"$"为结束符，返回单链表头指针 */
        linklist *head,*s;
        head=NULL;       /* 链表开始为空 */
        ch=getchar( );       /* 读入第一个结点的值 */
        while (ch!='$')
        { s=( linklist *)malloc(sizeof(linklist));      /* 生成新结点 */
```

（2 分）

```
        s→data=ch;        /* 将输入数据放入新结点的数据域中 */
        s→next=head;
        head=s;          /* 将新结点插入到表头上 */
        ch=getchar( );     /* 读入下一个结点的值 */                    （5 分）
      }
      return head;       /* 返回链表头指针 */
} /* CREATLISTF */
```

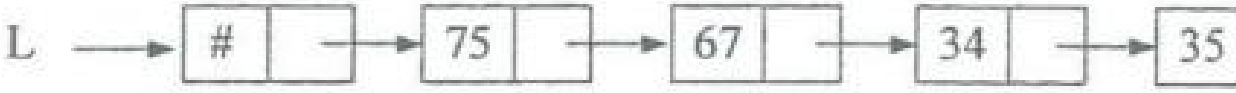
(2) 尾插法：

```
linklist *CREATLISTR()  /* 尾插法建立单链表，返回表头指针 */
    { char ch;
      linklist *head,*s,*r;
      head=NULL;              /* 链表初值为空 */
      r=NULL;                 /* 尾指针初值为空 */
      ch=getchar();           /* 读入第一个结点值 */
      while (ch!='$')         /* '$'为输入结束符 */
      { s=( linklist *)malloc(sizeof(linklist)); /* 生成新结点*s */
        s→data=ch;
        if (head==NULL) head=s;       /* 新结点*s 插入空表 */
        else r→next=s;          /* 非空表，新结点*s 插入到尾结点 */
        r=s;                    /* 尾指针 r 指向新的表尾 */
        ch=getchar();           /* 读入下一结点值 */
      }
      if (r!=NULL) r→next=NULL;    /* 对非空表，将尾结点的指针域置空 */
      return head;                 /* 返回单链表头指针 */
    }  /* CREATLISTR */
```

（3）带头结点的尾插法：

```
linklist *CREATLISTR1( )
/* 尾插法建立带头结点的单链表，*/
    { char ch;
      linklist *head,*s,*r;
      head=( linklist *)malloc(sizeof(linklist));  /* 生成头结点 head */
      r=head;                 /* 尾指针指向头结点 */
      ch=getchar( );
      while(ch!='$')    /* "$"为输入结束符 */
      { s=( linklist *)malloc(sizeof(linklist));    /* 生成新结点*s */
        s→data=ch;
```

1、设 L 为带头结点的单链表，试分析算法的输出结果。




```
void   Out_Linklist(LinkList*L){
       if(L->next!=NULL){
            Out_Linklist(L->next);
            if (L->next->data%2==1) (printf("%5d", L->next->data);
       }
}
```

线性表统计问题

2、假设单向循环链表的结点类型定义如下：

```
typedef   struct   node {
      char   data;
      struct   node*next;
} LinkList;
```

编写算法，统计带有头结点的头指针为 head 的非空单向循环链表中 data 域值为大写或小写字母的结点个数占结点总数的百分比。函数原型为：double count(LinkList* head);

```
double count(LinkList* head)
{
double x=0,n=0; LinkList* q=head->next; while(q!=head)
{
if((q->data>=65&&q->data<=90)|| (q->data>=97&&q->data<=122))
          //大写 65-90
          //小写 97-122
{
     x++;
}
n++;
q=q->next;
}
return x÷n;
}
```

2、A 是带头结点的的单链表，其数据元素是字符字母、数字字符、其他字符，将 A 表分成三个带头结点的循环单链表 A、B 和 C，其中 A 链表含有字符字母、B 链表含有数字字符、C 链表含有其它字符，要求利用原表空间。

```
void  one_to_three(linklist * L, linklist *A, *B, *C)
{ linklist *p=L->next; linklist *a, *b, *c;
  // p 为工作指针，指向 A 表的当前元素，r 为当前元素的后继指针，使表避免断开。
  //算法思想是取出当前元素，根据是字母、数字或其它符号，分别插入相应表中。
  b=B=(linklist *)malloc(sizeof(linklist));//申请空间，不判断溢出
  b->next=B->next=null;              // 准备循环链表的头结点
  c=C=(linklist *)malloc(sizeof(linklist));//申请空间，不判断溢出
  c->next=C->next=null;              // 准备循环链表的头结点
  while(p)
   {  // 用以记住后继结点
     if (p->data>=' a' &&p->data<=' z' ||p->data>=' A' && p->data<=' Z' )
        {a->next=p; a=p;}  // 将字母字符插入 A 表
      else if (p->data>=' 0' &&p->data<=' 9' )
            { b->next=p; b=p;}  // 将数字字符插入 B 表
         else { c->next=p; c=p;;}// 将其它符号插入 C 表
     p=p->next;                  //指向下一个后继结点
   }//while
a->next=A;
b->next=B;
c->next=C;
  }  // 算法结束
```

2.（5 分）已知有两个按元素值递增次序排列的线性表，均以不带头结点的单链表形式存储。请编写算法将这两个单链表归并为一个仍按元素值递增（非递减）排列的单链表，并要求利

用原来两个单链表的结点存放归并后的单链表。类型定义如下：

```
typedef struct LNode{
    DataType data;
    struct LNode *next;
}LNode, *LinkedList;

LinkedList CombineList(LinkedList la, LinkedList lb)
{
    LinkedList lc=NULL, lce=NULL;
    while (la && lb)
    {
        if (la->data<lb->data)
        {
            if (!lc) lc=la;
            else  lce->next=la;
            lce=la;
            la=la->next;
        }
        else {
            if (!lc) lc=lb;
            else  lce->next=lb;
            lce=lb;
            lb=lb->next;
        }
    }
    If (lc)
        if (la) lce->next=la;
        else lce->next=lb;
    else
        if (la) lc=la;
        else lc=lb;
```