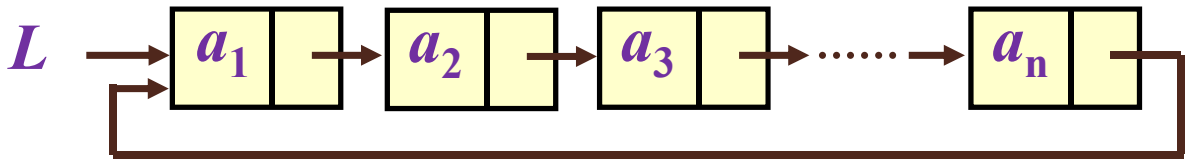
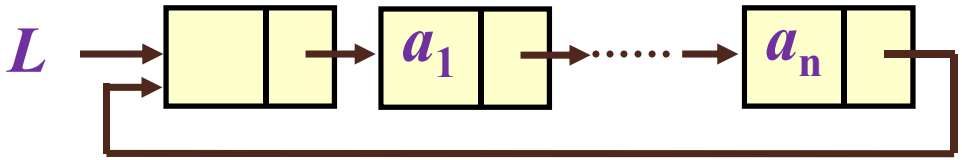
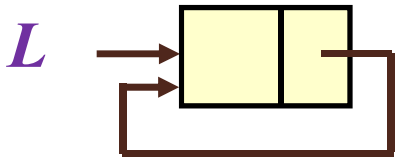
2.3.2 循环链表(Circular List)

- 循环单链表是单链表的变形。链表尾结点的 next指针不是 NULL，而是指向了单链表的前端。



- 为简化操作，在循环单链表中往往加入头结点。



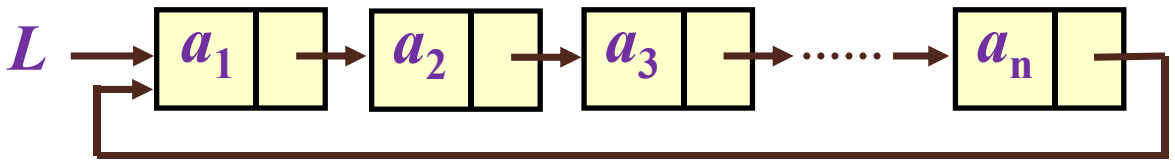


(空表的状态)

- 循环单链表的判空条件是：L->next == L。
- 循环单链表的特点是：只要知道表中某一结点的地址，就可搜寻到所有其他结点的地址。
- 在搜寻过程中，没有一个结点的 next 域为空。

```
for ( p = L->next;  p != L; p = p->next )
    ……;
```

- 循环单链表的所有操作的实现类似于单链表，差别在于检测到链尾，指针不为NULL，而是回到链头。



循环单链表的算法设计举例

（1）初始化线性表运算算法

创建一个空的循环单链表，它只有头结点，由*L*指向它。该结点的next域指向该头结点。

```
Status InitList_cl(LinkList &L)
{ // 操作结果：构造一个空的循环链表L
    // 产生头结点,并使L指向此头结点
    L = (LinkList)malloc(sizeof(LNode));
    if(!L)                          // 存储分配失败
        exit(OVERFLOW);
    L->next = L;                    // 指针域为空
    return OK;
}
```

（2）求线性表的长度算法

设置变量i作为计数器，i初值为0，p初始时指向第一个结点。然后沿next域逐个往下移动，每移动一次，i值增1。当p所指结点为头结点时这一过程结束，i之值即为表长。

```
int ListLength(LinkList L)
{
    // 初始条件：线性表L已存在。操作结果：返回L中数据元素个数
    int i = 0;
    LinkList  p = L->next;      // p指向第一个结点
    while(p != L) {             // 没到表尾
        i++;
        p = p->next;
    }
    return i;
}
```

循环单链表的算法设计示例

【示例-1】设计一个算法求一个循环单链表L中所有值为x的结点个数。

解：用指针p遍历循环单链表L的所有结点，用i（初值为0）统计值为x的结点个数。

```
int CountNode(LNode *L, ElemType x)
{  int i = 0;
   LNode *p=L->next;
   while (p != L)           //遍历整个循环链表
   {    if (p->data==x) i++;
        p=p->next;
   }
   return i;
}
```

【示例-2】有一个非递减有序的循环单链表L，设计一个算法删除其中所有值为x的结点，并分析算法的时间复杂度。

解：由于循环单链表L是非递减有序的，则所有值为x的结点必然是相邻的。

- 先找到第一个值为x的结点p，让pre指向其前驱结点。
- 然后通过pre结点删除p结点及其后面连续值为x的结点。

```
int Delallx(LNode *&L, ElemType x)
{
   LNode *pre=L,*p=L->next;          //pre指向p结点的前驱结点
   while (p!=L && p->data!=x)        //找第一个值为x的结点p
   {    pre=p;
        p=p->next;
   }
   if (p==L) return 0;                //没有找到值为x的结点返回0
   while (p!=L && p->data==x)         //删除所有值为x的结点
   {    pre->next=p->next;
        free(p);
        p=pre->next;
   }
   return 1;                          //成功删除返回1
}
```

【示例-3】编写一个程序求解约瑟夫（Joseph）问题。

有n个小孩围成一圈，给他们从1开始依次编号，从编号为1的小孩开始报数，数到第m个小孩出列，然后从出列的下一个小孩重新开始报数，数到第m个小孩又出列，…，如此反复直到所有的小孩全部出列为止，求整个出列序列。

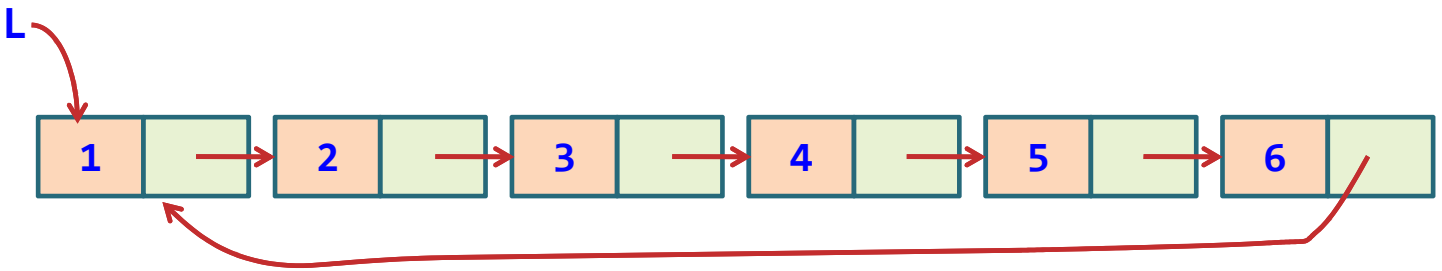
如当n=6，m=5时的出列序列是5，4，6，2，3，1。

解：（1）设计存储结构

本题采用循环单链表存放小孩圈，其结点类型如下：

```
typedef struct node
{  int no;                    //小孩编号
   struct node *next;         //指向下一个结点指针
} Child;
```

依本题操作，小孩圈循环单链表不带头结点，例如，n=6时的初始循环单链表如下图所示，L指向开始报数的小孩结点。



（2）设计基本运算算法

设计两个基本运算算法。由指定的*n*采用尾插法创建不带头结点的小孩圈循环单链表L的算法如下：

```
void CreateList(Child *&L, int n) //建立有n个结点的循环单链表
{  int i;  Child *p,*tc;      //tc指向新建循环单链表的尾结点
   L=(Child *)malloc(sizeof(Child));
   L->no=1;                   //先建立只有一个no为1结点的单链表
   tc=L;
   for (i=2;i<=n;i++)
   {   p=(Child *)malloc(sizeof(Child));
       p->no=i;               //建立一个存放编号i的结点
       tc->next=p; tc=p;      //将p结点链到末尾
   }
   tc->next=L;                //构成一个首结点为L的循环单链表
}
```

由指定的*n*和*m*输出约瑟夫序列的算法如下：

```
void Joseph(int n, int m)          //求解约瑟夫序列
{  int i,j;  Child *L,*p,*q;
   CreateList(L,n);

   for (i=1;i<=n;i++)     //出列n个小孩
   {    p=L; j=1;
        while (j<m-1)     //从L结点开始报数，报到第m-1个结点
        {  j++;           //报数递增
           p=p->next;     //移到下一个结点
        }
        q=p->next;              //q指向第m个结点
        printf("%d ",q->no);    //该结点出列
        p->next=q->next;        //删除q结点
        free(q);                //释放其空间
        L=p->next;              //从下一个结点重新开始
   }
}
```

（3）设计主函数

设计如下主函数求解n=6，m=5的约瑟夫序列：

```
void main()
{   int n=6,m=5;
    printf("n=%d,m=%d的约瑟夫序列:",n,m);
    Joseph(n,m);
    printf("\n");
}
```

END