线性表的数据元素之间有一对一的逻辑关系

顺序存储：顺序表；链式存储：链表

① 顺序表：基于数组，用一组地址连续的存储单元，依次存储线性表中各种元素

②链表：

Array：优点：易访问，元素集中

缺点：需初始化确定长度、难变长度、难增删元素

Linked List：每个元素都有一个节点，每个节点都指向下一个，最后一个的节点是null

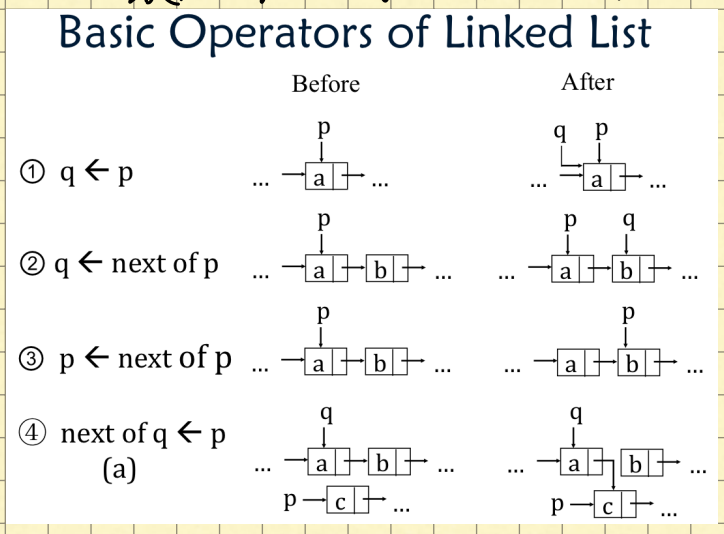
在数组中，元素在内存中是连续储存的.

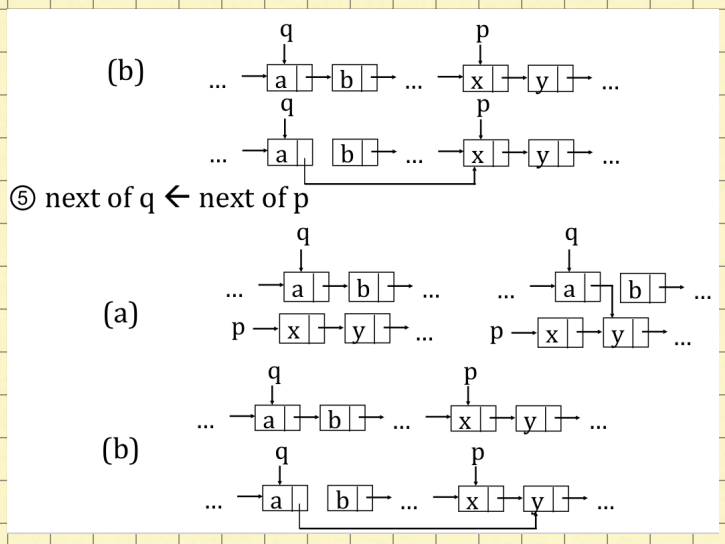
在链表中，元素都是离散的对象，在内存中不一定连续.

优点：易变长度、易增删

缺点：难访问，元素不集中

Basic Operators of Linked List





遍历：

Algorithm: traverse(A):

```
1. if (A=NULL)
2.       return
3. else
4.     print A.value
5.     traverse(A.next)
```

Algorithm: traverseIteration(A):

```
1. node trav ← A
2. While (trav != NULL)
3.       print trav.value
4.       trav ← trav.next
```

插入：

- **Problem:** insert node q in Linked List A at Position i
- **Algorithm:** insertNode(A, node q, i):

```
1. a ← 0, node p ← A,
2. while (i-1 > a)
3.       p ← p.next
4.       a ← a + 1
5. tmp ← p.next
6. p.next ← q
7. q.next ← tmp
8. return A
```

- Time Complexity: **O(n)**
- Space Complexity: **O(1)**

删除：

- **Problem:** delete node in Linked List A at Position i
- **Algorithm:** deleteNode(A, i):

```
1. a ← 0, node p ← A,
2. while (i-1 > a)
3.       p ← p.next
4.       a ← a + 1
5. p.next ← p.next.next
6. return A
```

- Time Complexity: **O(n)**
- Space Complexity: **O(1)**

查询

- **Problem:** Find value x in Linked List A
- **Algorithm:** findValue(A, x):

```
1. a ← 0, node p ← A,
2. while (p!=NULL)
4.       if (x = p.value)
5.             return p
6.       p ← p.next
7. return -1
```

- Time Complexity: **O(n)**
- Space Complexity: **O(1)**

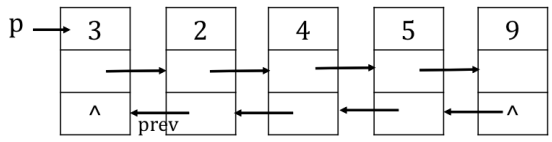
更新

- **Problem:** Update nodes with value x to y in Linked List A
- **Algorithm:** updateNodes(A, x):

```
1. a ← 0, node p ← A,
2. while (p!=NULL)
4.       if (x = p.value)
5.             p.value ← y
6.       p ← p.next
7. return A
```

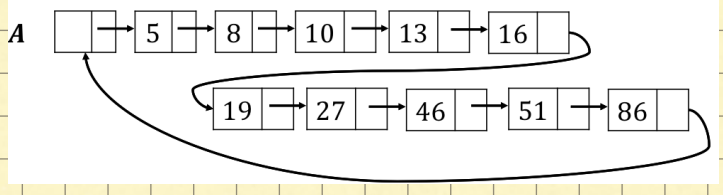
- Time Complexity: **O(n)**
- Space Complexity: **O(1)**

Double linked list



- add a prev reference to each node: refers to the previous node
- allow us to "back up" from a given node

Circular linked list



block-based linked list