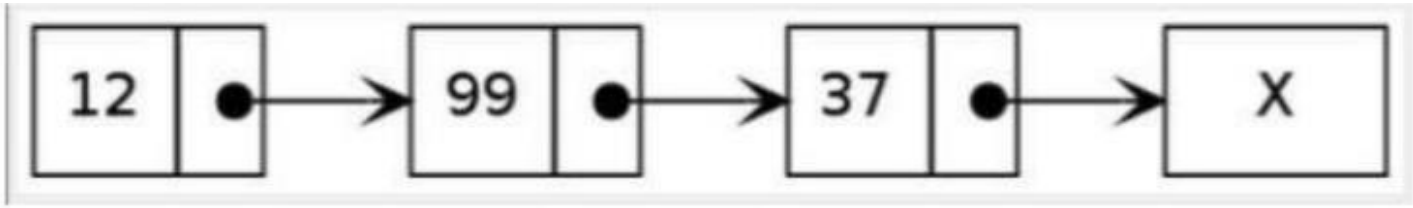
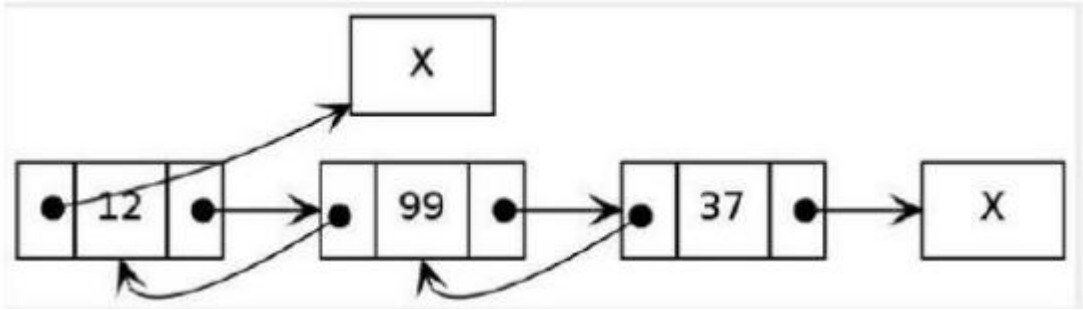
Linked lists – what are they?

- A linked list is a data structure which also stores items in order, accessed by index, without gaps.
- The items have contiguous indices, but it does not use contiguous memory - an element can be stored anywhere in memory, regardless of where the element with the index one higher or one lower is stored.
- Each element stores a pointer to the next element - the "link".
- Last element points to null.
- First item is the "head".
- Access the ith element by following i links from the head.

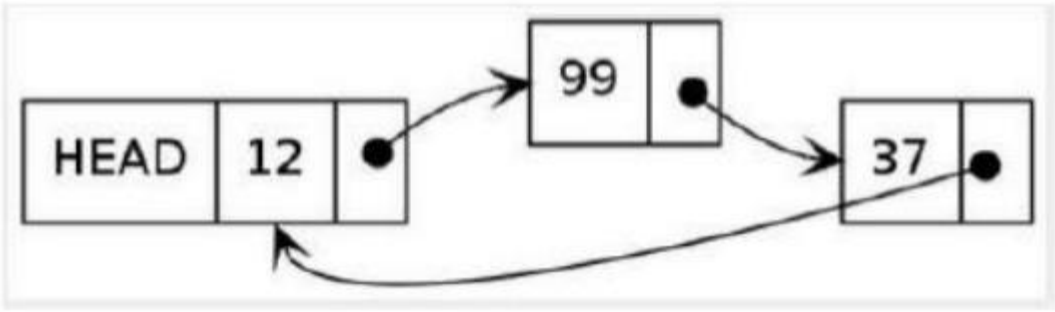


Double linked lists and circular lists

- A doubly linked list also stores a pointer to the previous element of the list.



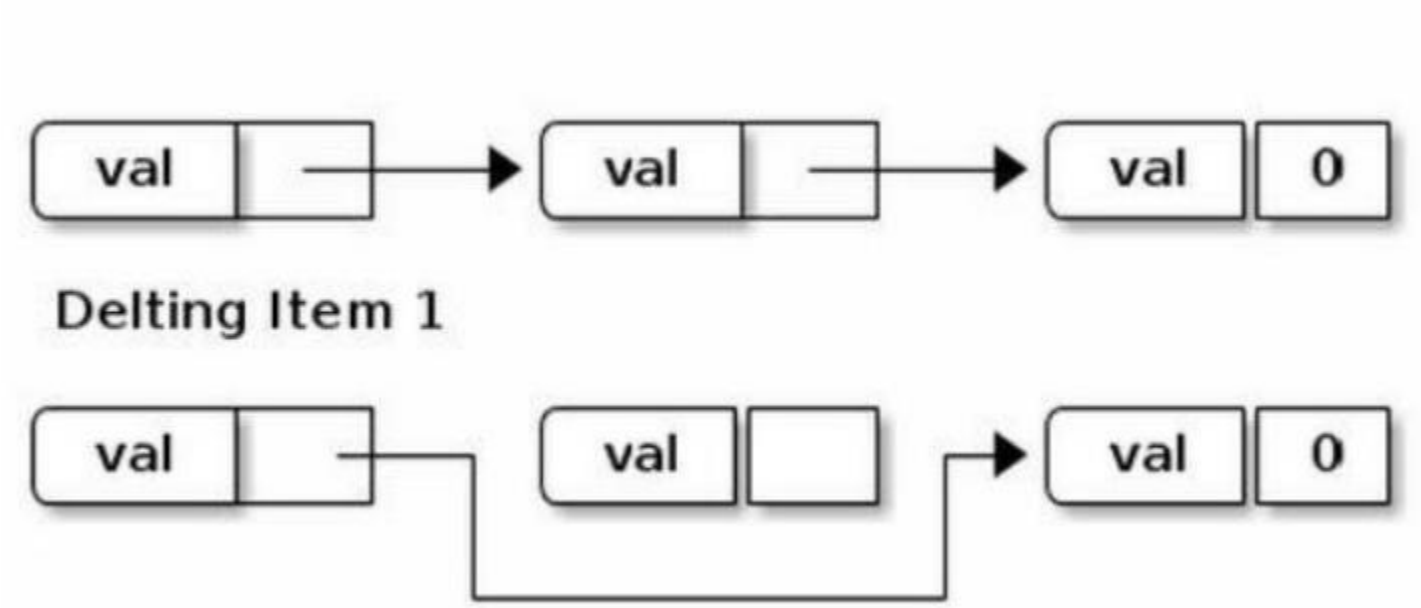
- A circular linked list stores a pointer from the last element to the first element.



Double linked lists and circular lists

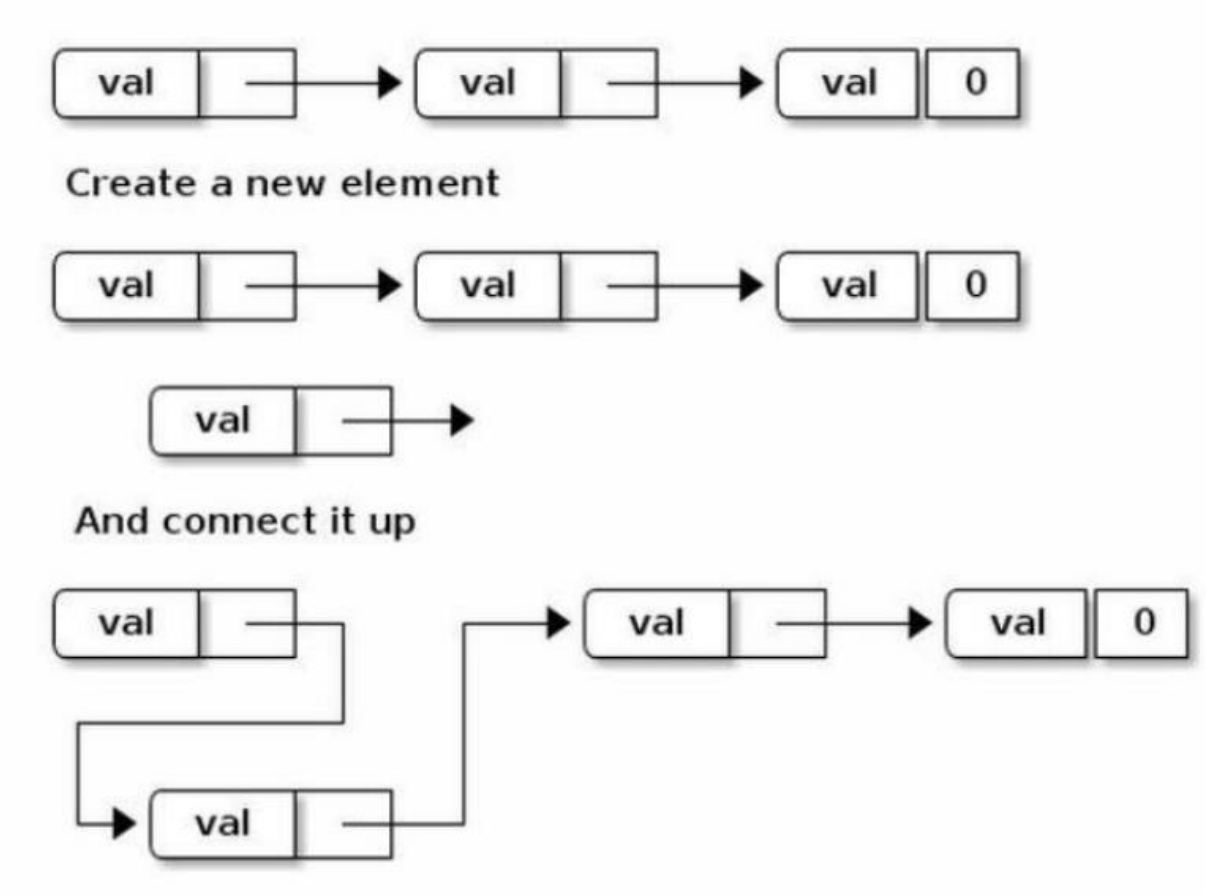
- To reach a given item, the list is traversed, beginning with the head.
- Is this the item you want?
- If not, follow the link and repeat.
- Deleting an item is achieved by changing the link from the previous item to point to the one after the one we want deleted.
- Inserting is done by creating a new item and then changing the link from the item we want it to follow to point to it, and making the new item's link point to the item that previously followed the item that now links to it.
- These operations make more sense with a diagram and some notation.

Deleting an element



- If your language doesn't perform automatic garbage collection, you must now erase/delete the disconnected element.

Inserting an element



Efficiency

- Linked lists have the opposite characteristics of an array.
- Access to a particular element is slow because we have to work our way through each element of the list to get there, it's O(n).
- However, insertion and deletion is fast (once we are at the element we want to insert at) because we merely have to change which element the pointer is pointing to, this is O(1).

Creating a linked list

- We can consider a doubly linked list nodeN to have three attributes:
  - N.value - the data item stored by the node.
  - N.previous - a pointer to the previous node in the list.
  - N.next - a pointer to the next node in the list.
- We can also define a list object, L, which contains two attributes:
  - L.head - the first node in the list.
  - L.tail - the last node in the list.
- In the definitions of object functions, the first parameter is still listed as the object in question:
  - Like "self" in Python.
  - Like the implicit "this" in C++.

Inserting and deleting

- We can define an operation insert which will place the node X immediately after the node N in the list L using the following pseudocode, for the first item we can just use N = (A NULL pointer).
- Note that N is a node, not an index, just like X.

```
LIST-INSERT(L, N, X)
if N ≠ ∅
    X.next ← N.next
    N.next ← X
    X.prev ← N
    if X.next ≠ ∅
        X.next.prev ← x
if L.head = ∅
    L.head ← L.tail ← X
    X.prev ← X.next ← ∅
else if L.tail = N
    L.tail ← X
```

```
LIST-REMOVE(L, N)
if N.prev ≠ ∅
    N.prev.next ← N.next
else
    L.head ← N.next
if N.next ≠ ∅
    N.next.prev ← N.prev
else
    L.tail ← N.prev
```

Implementation

Python

C++

Summary

- Arrays are:
  - Very efficient for access and modification of a specific element.
  - Not efficient for insertion and removal.
- Linked lists are:
  - Not efficient for finding, access and modification of a specific element.
  - Very efficient for insertion and removal.