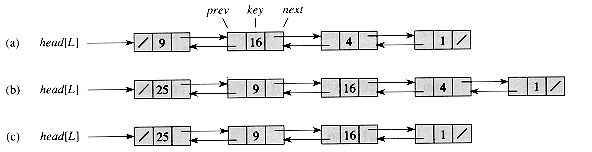
**Linked lists**

A ***linked list*** is a data structure in which the objects are arranged in a linear order. Unlike an array, however, in which the linear order is determined by the array indices, the order in a linked list is determined by a pointer in each object. In linked list me can insert or delete an object in arbitrary position. The order in linked list is determined by a pointer in each position.

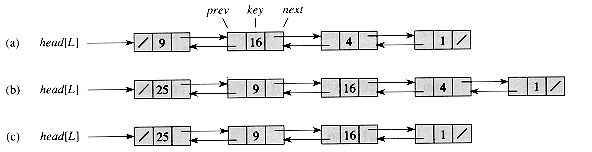


As shown in Figure, each element of a doubly linked list L is an object with a key field and two other pointer fields: **next** and **prev**. The object may also contain other satellite data. Given an element x in the list, **next[x]** points to its successor in the linked list, and **prev[x]** points to its predecessor. If **prev[x] = NIL**, the element x has no predecessor and is therefore the first element, or **head**, of the list. If

**next [x] = NIL**, the element x has no successor and is therefore the last element, or **tail**, of the list. An attribute **head[L]** points to the first element of the list. If

**head[L] = NIL**, the list is empty.

A list may be either **singly linked** or **doubly linked**. If a list is **singly linked**, we omit the **prev pointer** in each element.



**Figure (a) A doubly linked list L representing the dynamic set {9, 16, 4, 1}. Each element in the list is an object with fields for the key and pointers (shown by arrows) to the next and previous objects. The next field of the tail and the prev field of the head are NIL, indicated by a diagonal slash. The attribute head[L] points to the head.**

**(b) Following the execution of LIST-INSERT(L, x), where key[x] = 25, the linked list has a new object with key 25 as the new head. This new object points to the old head with key 9.**

**(c) The result of the subsequent call LIST-DELETE(L, x), where x points to the object with key 4.**

**Searching a linked list**

The procedure LIST-SEARCH(L, k) finds the first element with key k in list L by a simple linear search, returning **a pointer** to this element. If no object with key k appears in the list, then NIL is returned. For the linked list in Figure (a), the call LIST-SEARCH(L, 4) returns **a pointer** to the third element, and the call LIST-SEARCH (L, 7) returns NIL.

LIST-SEARCH (*L*, *k*)

1 *x* http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/arrlt12.gif head[L]

2 **while** *x* http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/noteq.gif NIL and *key*[*x*] http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/noteq.gif *k*

3 **do** *x* http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/arrlt12.gif *next*[*x*]

4 **return** *x*

**Inserting into a linked list**

Given an element *x* whose *key* field has already been set, the LIST-INSERT procedure "splices" *x* onto the **front** of the linked list, as shown in Figure (b).

LIST-INSERT(*L, x*)

1 *next*[*x*] http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/arrlt12.gif *head*[*L*]

2 **if** *head*[*L*] http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/noteq.gif NIL

3 **then** *prev*[*head*[*L*]] http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/arrlt12.gif *x*

4 *head*[*L*] http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/arrlt12.gif *x*

5 *prev*[*x*] http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/arrlt12.gif NIL

**Deleting from a linked list**

The procedure LIST-DELETE removes an element *x* from a linked list *L*. It must be given a pointer to *x*, and it then "splices" *x* out of the list by updating pointers. If we wish to delete an element with a given key, we must first call LIST-SEARCH to retrieve a pointer to the element.

LIST-DELETE(*L*, *x*)

1 **if** *prev*[*x*] http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/noteq.gif NIL

2 **then** *next*[*prev*[*x*]] http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/arrlt12.gif *next*[*x*]

3 **else** *head*[*L*] http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/arrlt12.gif *next*[*x*]

4 **if** *next*[*x*] http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/noteq.gif NIL

5 **then** *prev*[*next*[*x*]] http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/arrlt12.gif *prev*[*x*]

Figure (c) shows how an element is deleted from a linked list.