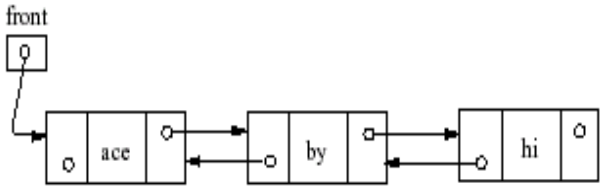
Doubly Linked List or Two way Linked List

- A doubly linked list is a linked list in which every node as a next pointer and a back pointer
- Every node (except the last node) contains the address of the next node, and every node (except the first node) contains the address of the previous node.
- A doubly linked list can be traversed in either direction

Doubly-linked lists

Each element keeps both a frontwards and a backwards reference.



- The node is divided in to 3 parts
  1. Information field
  2. Forward Link which points to the next node
  3. Backward Link which points to the previous node
- The starting address or the address of first node is stored in START / FIRST pointer
- Another pointer can be used to traverse Doubly LL from end. This pointer is called END or LAST

Operations on DLL

1. Traversing DLL
2. Searching an item in DLL
3. Insertion
4. Deletion

Traversing and Searching

- Traversing is same as that in Single LL except that there are now 2 ways to traverse the List

  Starting from 1st element

  i. In this case START will be used to initialize PTR
  ii. And PTR := FLINK(PTR) to move to next element

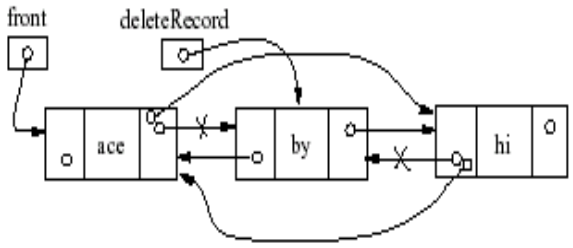
  Starting from last element

i. In this case END will used to initialize PTR
ii. And PTR := BLINK(PTR) to move to next element

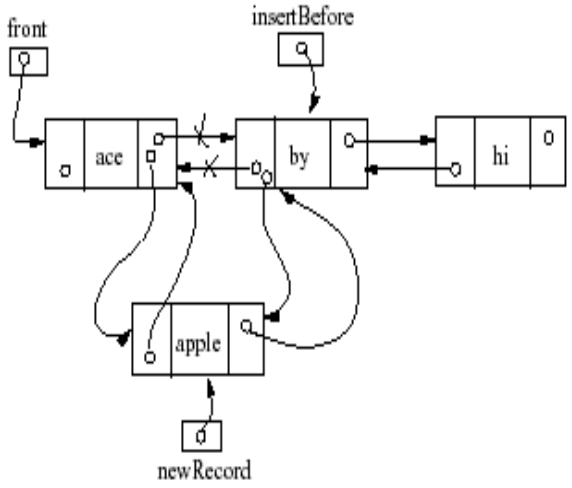
- Searching is same as in traversing and based on traversing scheme condition will be applied

When traversing a doubly-linked list to prepare for insertion or deletion, there is no need to use both a current and previous reference: each node has a built-in reference to the previous element.

Deletion with a doubly-linked list:



Insertion with a doubly-linked list:



Header Nodes

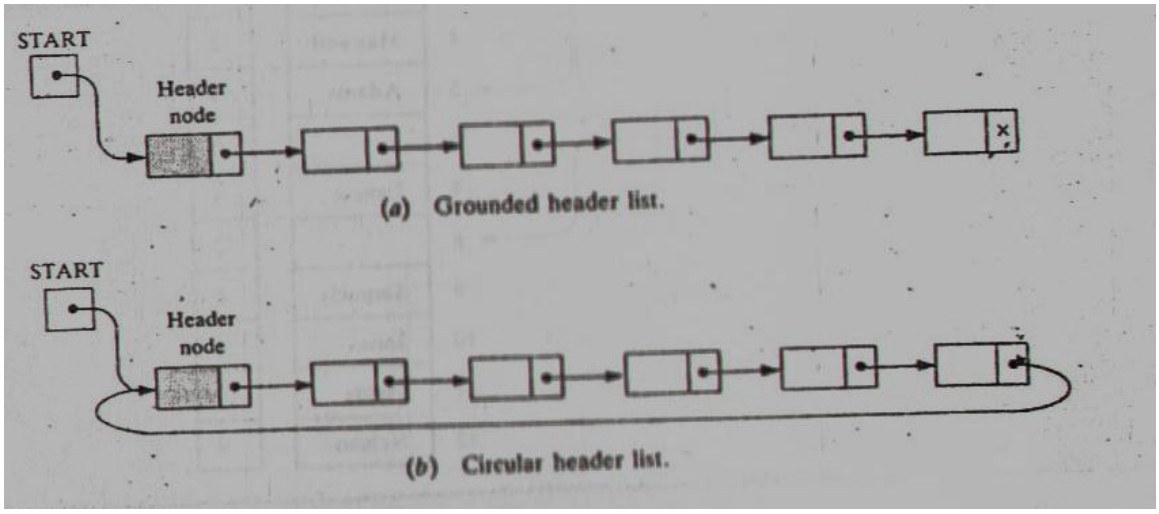
- One way to simplify insertion and deletion is never to insert an item before the first or after the last item and never to delete the first node
- You can set a header node at the beginning of the list containing a value smaller than the smallest value in the data set
- You can set a trailer node at the end of the list containing a value larger than the largest value in the data set.
- These two nodes, header and trailer, serve merely to simplify the insertion and deletion algorithms and are not part of the actual list.
- The actual list is between these two nodes.

5.9 HEADER LINKED LISTS

A *header* linked list is a linked list which always contains a special node, called the *header node*, at the beginning of the list. The following are two kinds of widely used header lists:

(1) A *grounded header list* is a header list where the last node contains the null pointer. (The term "grounded" comes from the fact that many texts use the electrical ground symbol to indicate the null pointer.)

(2) A *circular header list* is a header list where the last node points back to the header node.



Algorithm 5.11: (Traversing a Circular Header List) Let LIST be a circular header list in memory. This algorithm traverses LIST, applying an operation PROCESS to each node of LIST.

1. Set PTR := LINK[START]. [Initializes the pointer PTR.]
2. Repeat Steps 3 and 4 while PTR ≠ START:
3. Apply PROCESS to INFO[PTR].
4. Set PTR := LINK[PTR]. [PTR now points to the next node.]
   [End of Step 2 loop.]
5. Exit.

Algorithm 5.12: SRCHHL(INFO, LINK, START, ITEM, LOC)
LIST is a circular header list in memory. This algorithm finds the location LOC of the node where ITEM first appears in LIST or sets LOC = NULL.

1. Set PTR := LINK[START].
2. Repeat while INFO[PTR] ≠ ITEM and PTR ≠ START:
   Set PTR := LINK[PTR]. [PTR now points to the next node.]
   [End of loop.]
3. If INFO[PTR] = ITEM, then:
   Set LOC := PTR.
   Else:
   Set LOC := NULL.
   [End of If structure.]
4. Exit.