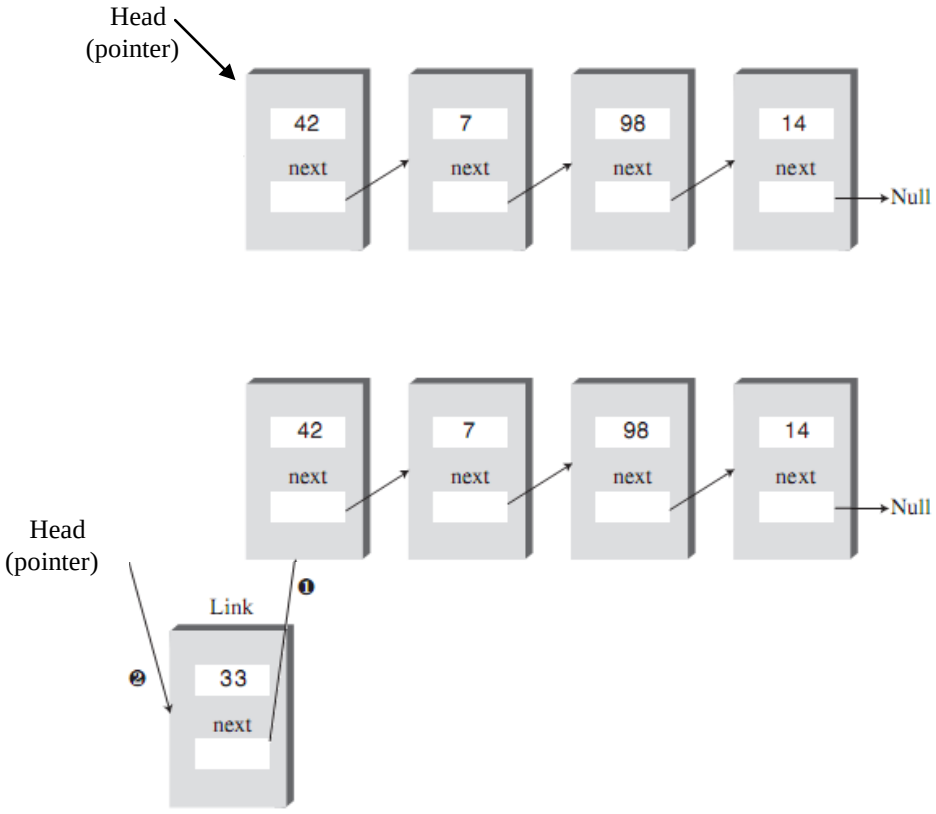
Operations on Singly (Simply) Linked Lists

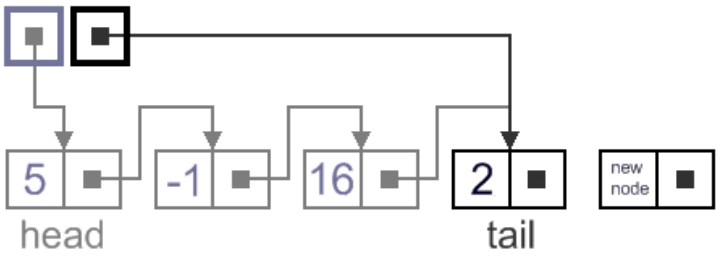
InsertFirst

The insertFirst() method of LinkList inserts a new link at the **beginning of the list**. To insert the new link, we need only set the next field in the newly created link to point to the old first link and then change first so it points to the newly created link. This situation is shown in Figure below. The insertion it can be done in the following steps:

- Create a new node.
- Update the **next** link of a new node, to point to the **current head node**.
- Update **head** link to point to the **new node** (i.e. make the new node to be the first node).

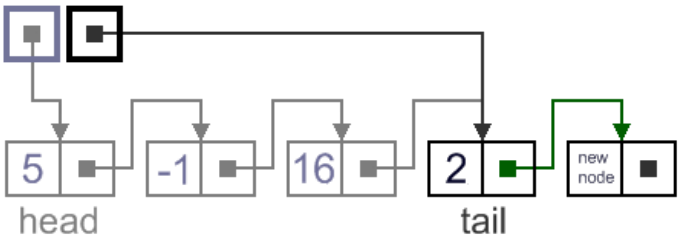


Add last

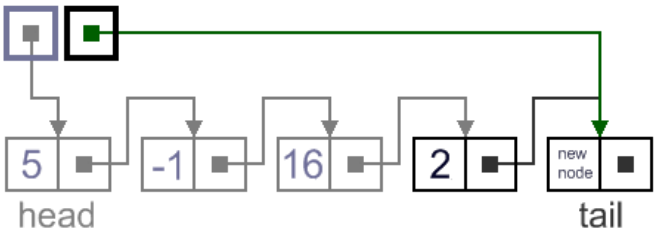


It can be done in the following steps:

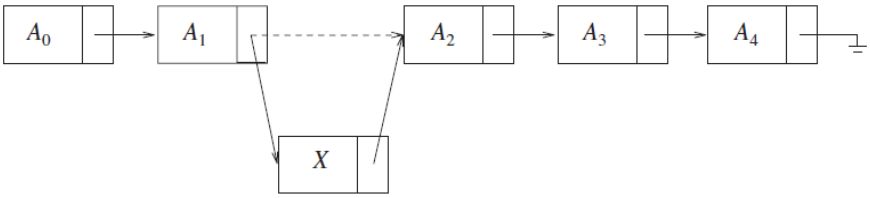
- Create a new node.
- Set the **next** link of a new node to **null**.
- Access to the last node of linked list (it is pointed by tail).
- Update the **next** link of the current tail node, to point to the new node.

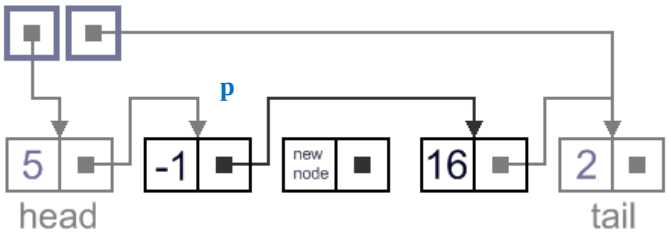


- Update **tail** link to point to the new node(make the new last node pointed by tail).



Inserted Between Two Nodes





Such an insert can be done in the following steps:

- ❖ Create a new node.
- ❖ Determine the position of insertion.
- ❖ Access to the node which is previous to the insertion position (it is pointed by **p**).
- ❖ If previous node (p) is null then *// means position =1*

  make a new node is the first node (call addfirst).

  Else

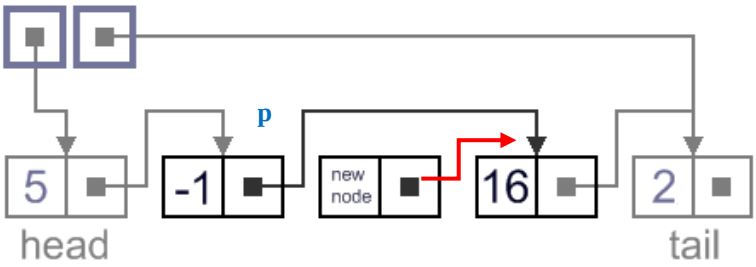
  If previous node is the tail of linked list then *// means position =n+1*

  make a new node is the last node (call addlast).

  Else

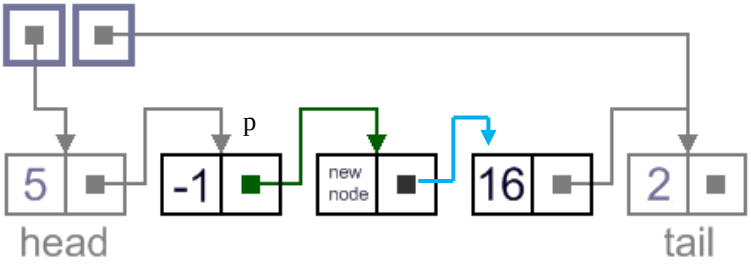
  - Update link of the **new node**, to point where the p.next points

    (newnode.next = p.next).



  - Update link of the **"previous"** node, to point to the **new node**

    (p.next = newnode).



Singly-linked list, Removal (deletion) operation.

There are **four cases**, which can occur while **removing** the node. These cases are similar to the cases in add operation. We have the same four situations, but the order of algorithm actions is opposite. Notice, that removal algorithm includes the **disposal** of the deleted node.

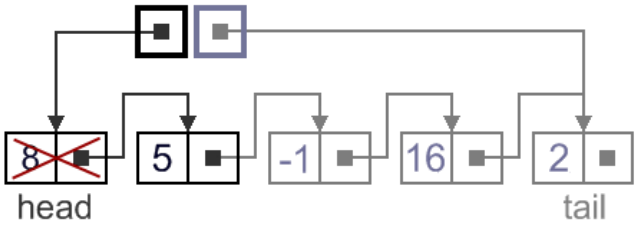
List has only one node

When list has only one node, which is indicated by the condition, that the **head points** to the **same node as the tail**, the removal is quite simple. Algorithm disposes the node, pointed by head (or tail) and **sets both head and tail to NULL**. This case can be done using removefirst method as:

head = head . next This sets head to null.

Here, you need to set the tail to null as: **tail = head** (tail becomes null).

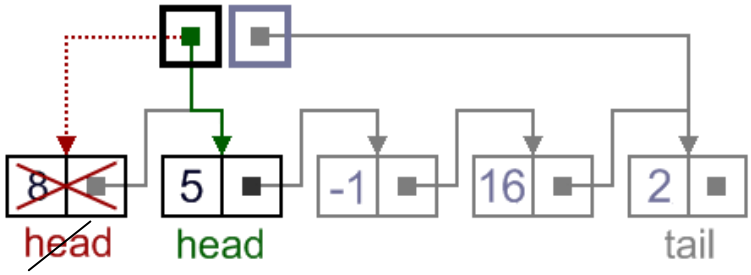
Remove first



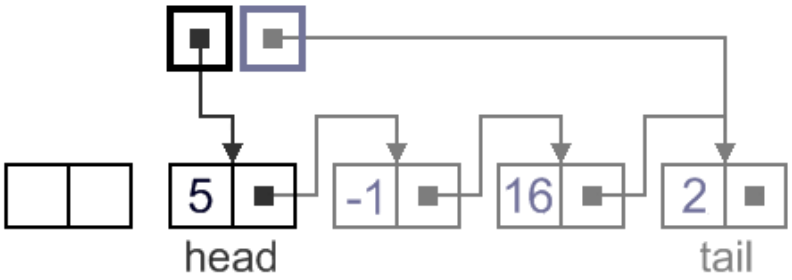
In this case, first node (current head node) is removed from the list. It can be done in two steps:

- Update **head link** to point to the node, next to the head.

head = head . next

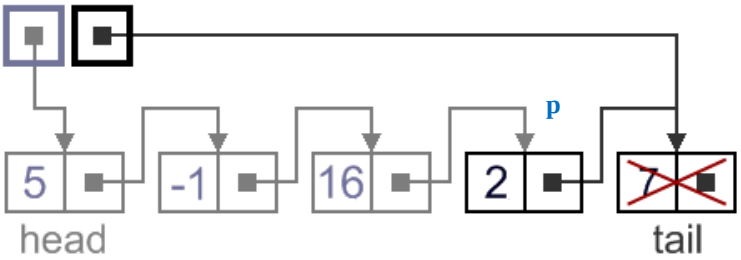


- Dispose removed node.



Remove last

In this case, last node (**current tail** node) is removed from the list. This operation is more tricky, than removing the first node, **because algorithm should** firstly **find a node**, which is **previous** to the tail (such as pointed by **p**).



It can be done in **the following steps**:

❖ Travel the nodes of linked list from beginning (head) to access the previous node (pointed with **p**) of last node. This can be done by one of the following:

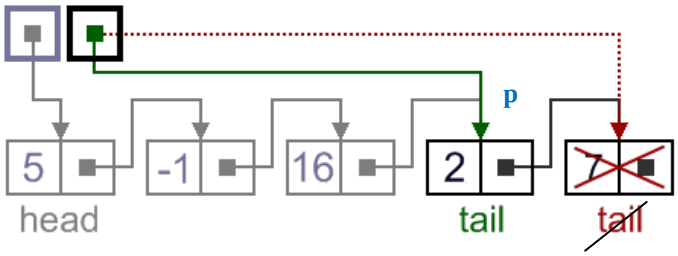
```
p = head
while p . next . next != null
    p = p . next
```

or

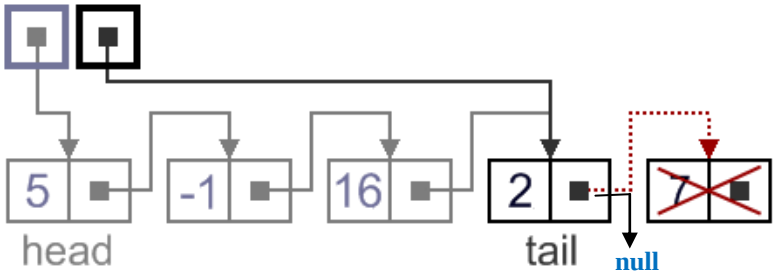
```
p = head
tail = p . next
while tail . next != null
    p = tail
    tail = tail . next
```

❖ Update **tail link** to point to the node that is before the last node (**p** node) as:

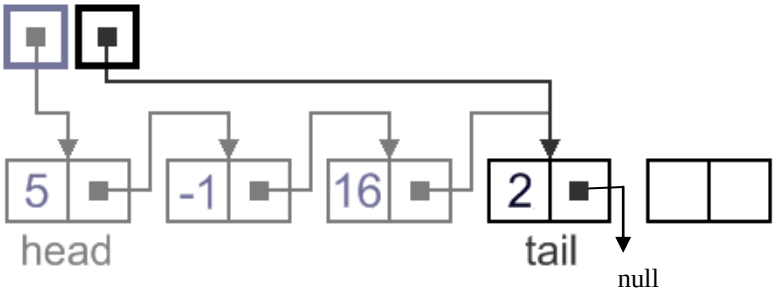
Tail = p



❖ Set **next link** of the new tail to **NULL**. (**tail . next = null**)

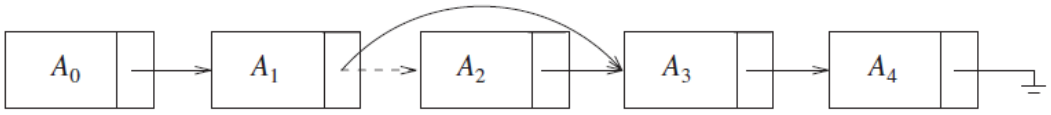


❖ Dispose removed node.

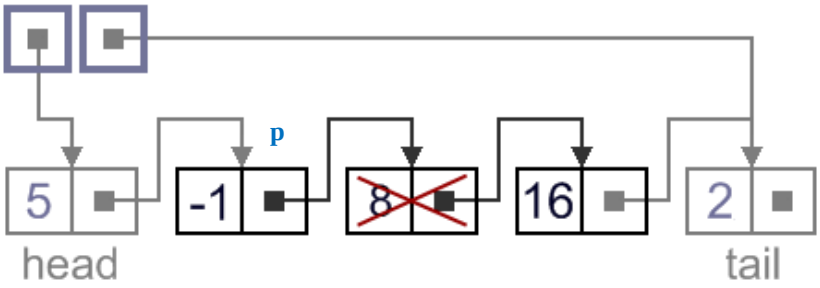


Remove Next

The remove method can be executed in **one next reference change**. The bellow figure shows the result of deleting the third element in the original list.



Such a removal can be done in **the following steps** :



- ❖ Determine the position of deletion.
- ❖ Access to the node which is previous to the deletion position (it is pointed by **p**).
- ❖ If previous node (p) is null then *// means position =1*

call removefirst

Else

If the next node of previous is tail (i.e. p . next = tail) then *// i.e. position =n*

call removelast to make :

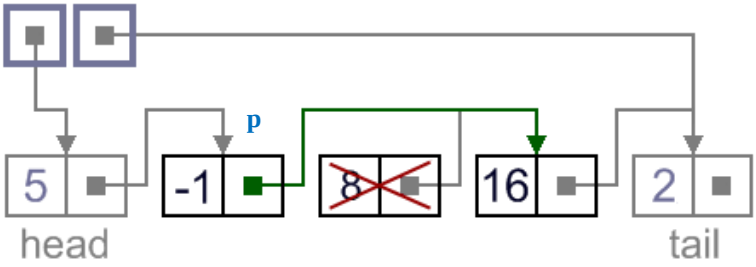
tail = p
tail . next = null

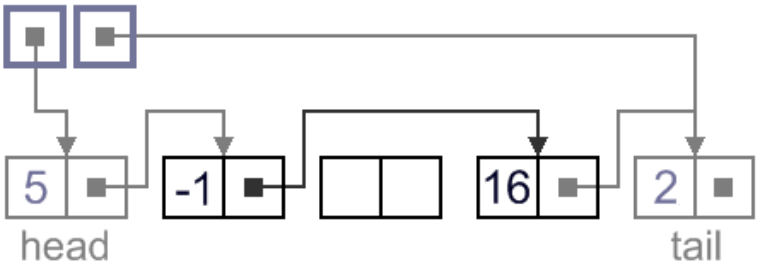
Else

- Update **next link** of the **previous node**, to point to the **next node** that is relative to the removed node.

(p . next = p . next . next).

- Dispose removed node.





H.w. Finding and Deleting Specified Links

Write a program that searches a linked list for a **data item** (without depending on position) with a specified key value and deletes **any** item with a specified key value.