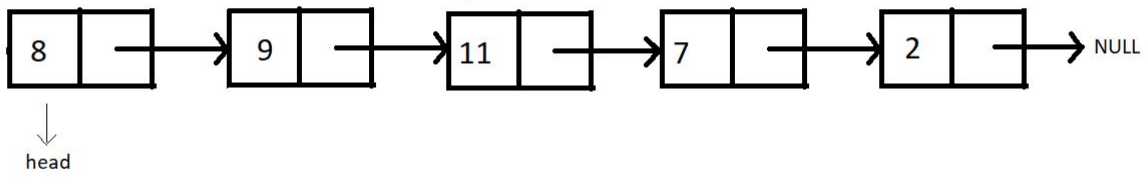
Deletion in a Linked List | Deleting a node from Linked List Data Structure

codewithharry.com/videos/data-structures-and-algorithms-in-hindi-17

In the last two tutorials, we got to see how one can insert a node in a linked list. Today, we'll learn how to delete a node at some position. It will draw quite similarities with inserting a node, so this might be easy to you as well.

Inserting in a linked list:

Consider the following Linked List:



Insertion in this list can be divided into the following categories:

Case 1: Deleting the first node.

Case 2: Deleting the node at the index.

Case 3: Deleting the last node.

Case 4: Deleting the first node with a given value.

For deletion, following any of the above-mentioned cases, we would just need to free that extra node left after we disconnect it from the list. Before that, we overwrite the current connection and make new connections. And that is how we delete a node from our desired place.

Syntax for freeing a node:

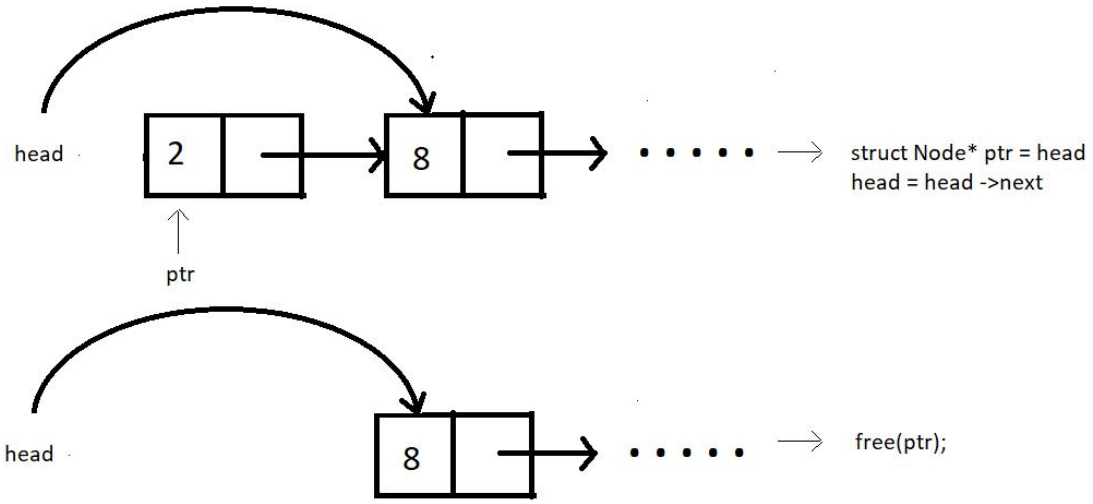
```
free(ptr);
```

The above syntax will free this node, that is, remove its reserved location in the heap.

Now, let's begin with each of these cases of insertion.

Case 1: Insert at the beginning:

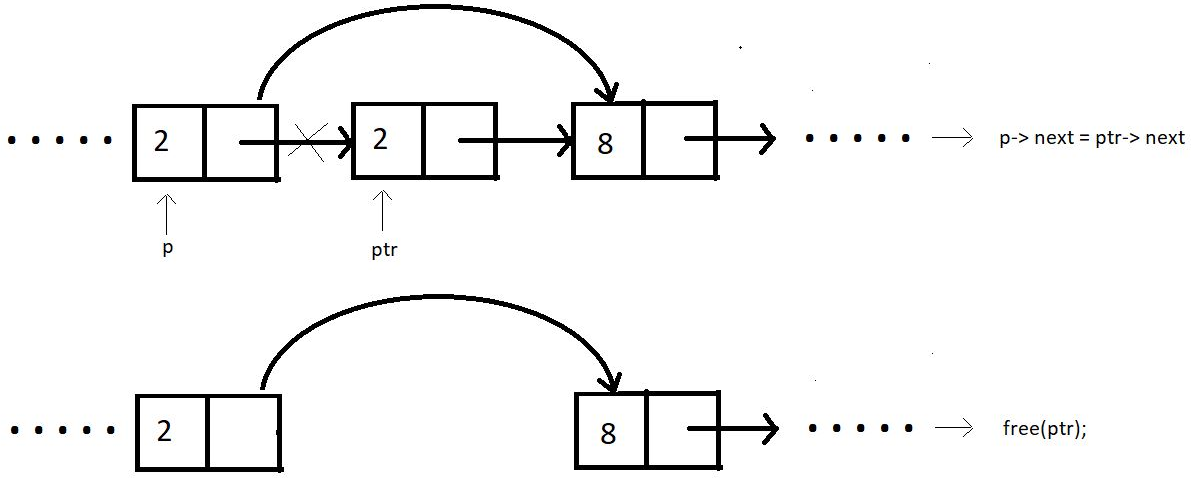
In order to delete the node at the beginning, we would need to have the head pointer pointing to the node second to the head node, that is, head-> next. And we would simply free the node that's left.



Case 2: Deleting at some index in between:

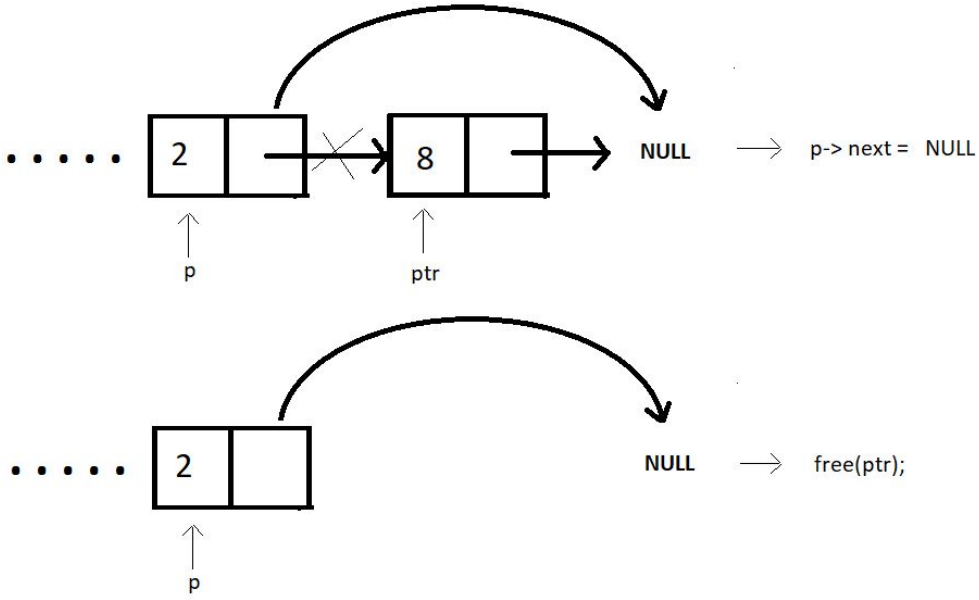
Assuming index starts from 0, we can delete an element from index i>0 as follows:

1. Bring a temporary pointer p pointing to the node before the element you want to delete in the linked list.
2. Since we want to delete between 2 and 8, we bring pointer p to 2.
3. Assuming ptr points at the element we want to delete.
4. We make pointer p point to the next node after pointer ptr skipping ptr.
5. We can now free the pointer skipped.



Case 3: Deleting at the end:

In order to delete an element at the end of the linked list, we bring a temporary pointer *ptr* to the last element. And a pointer *p* to the second last. We make the second last element to point at NULL. And we free the pointer *ptr*.

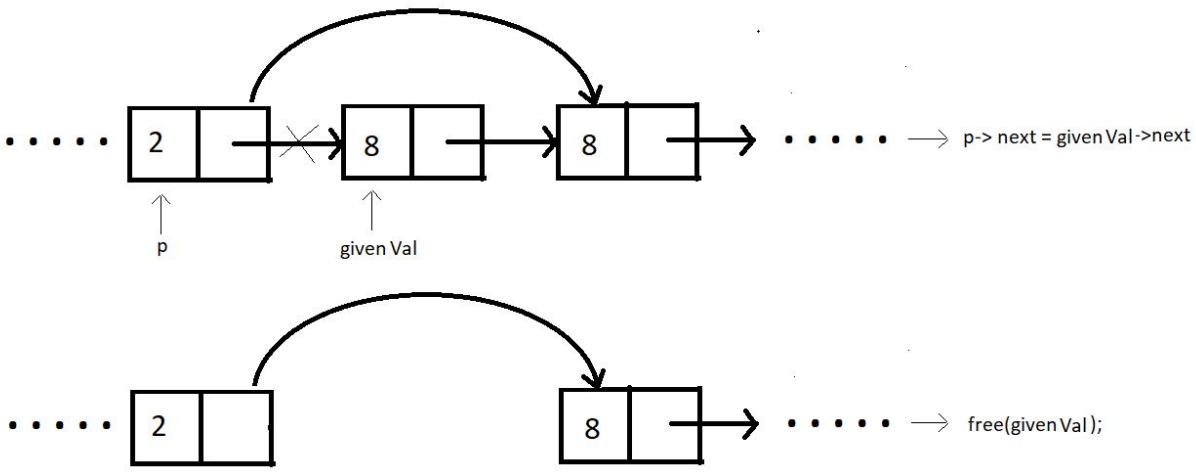


Case 4: Delete the first node with a given value:

Similar to the other cases, ptr can be deleted for a given value as well by following few steps:

1. p->next = givenVal-> next;
2. free(givenVal);

Since, the value 8 comes twice in the list, this function will be made to delete only the first occurrence.



Learning about the time complexity while deleting these nodes, we found that deleting the element at the beginning completes in a constant time, i.e O(1). Deleting at any index in between is no big deal either, it just needs the pointer ptr to reach the node to be deleted, causing it to follow O(n). And the same goes with case 3 and case 4. We have to traverse through the list to reach that desired position.

Before we move to coding these deletion methods, I want you all to try yourselves first. We'll see them in our next tutorial. I hope this was easy for you to understand following the fact that we have completed insertion.Thank you for being with me throughout. I hope you enjoyed the tutorial. If you appreciate my work, please let your friends know about this course too. Don't forget to download the notes from the link below. If you haven't checked out the whole playlist yet, move on to codewithharry.com or my YouTube channel to access it. See you all in the next tutorial where we'll try to code these deletion methods. Till then keep learning.

Download Notes Here