Delete given node in a Linked List : O(1) approach

Problem Statement: Write a function to **delete a node** in a singly-linked list. You will **not** be given access to the head of the list instead, you will be given access to **the node to be deleted** directly. It is **guaranteed** that the node to be deleted is **not a tail node** in the list.

Examples:

```
Example 1:

Input:

 Linked list: [1,4,2,3]

       Node = 2

Output:

Linked list: [1,4,3]

Explanation: Access is given to node 2. After deleting nodes, the linked list
will be modified to [1,4,3].



Example 2:

Input:

 Linked list: [1,2,3,4]

       Node = 1

Output: Linked list: [2,3,4]

Explanation:

 Access is given to node 1. After deleting nodes, the linked list will be
modified to [2,3,4].
```

Solution:

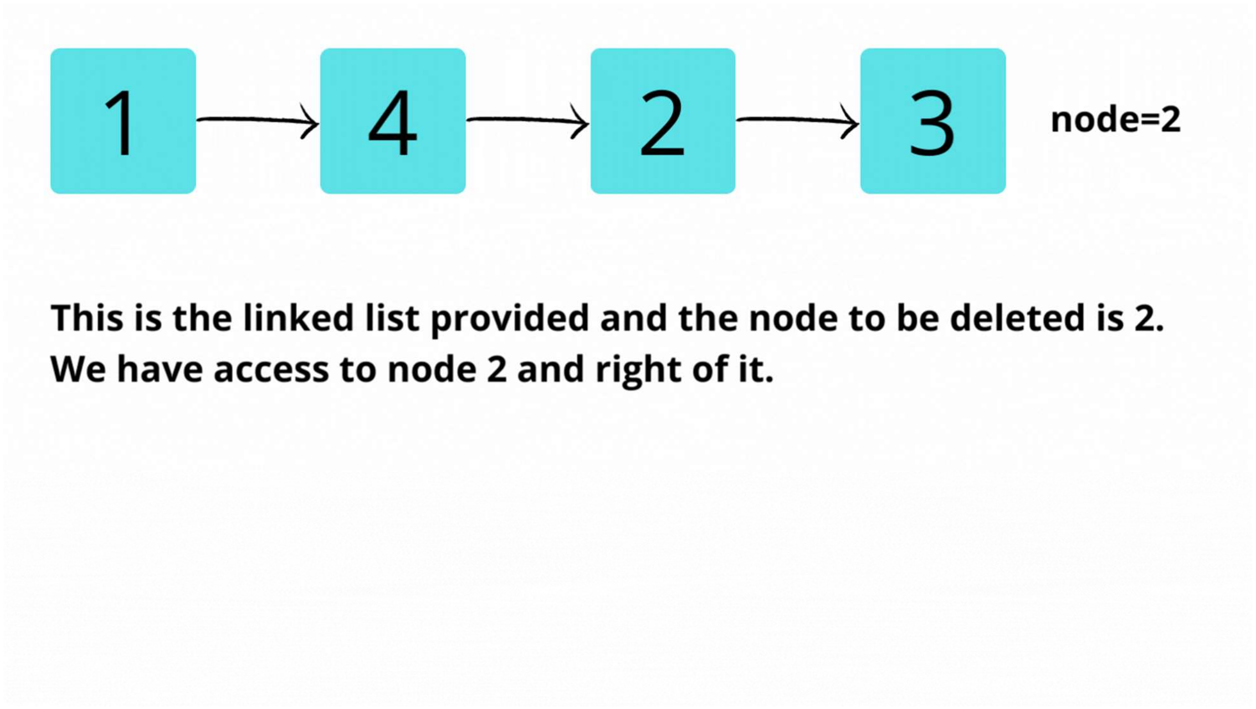
***Disclaimer**: Don't jump directly to the solution, try it out yourself first.*

Approach:

We are given access to nodes that we have to delete from the linked list. So, whatever operation we want to do in the linked list, we can operate in the right part of the linked list from the node to be deleted.

The approach is to copy the next node's value in the deleted node. Then, link node to next of next node. This does not delete that node but indirectly it removes that node from the linked list.

Dry Run:


1
4
2
3
node=2
This is the linked list provided and the node to be deleted is 2.
We have access to node 2 and right of it.

Code:

- C++ Code
- Java Code
- Python Code

```
#include<iostream>
using namespace std;
```

```cpp
class node {
    public:
        int num;
        node* next;
        node(int a) {
            num = a;
            next = NULL;
        }
};
//function to insert node at the end of the list
void insertNode(node* &head,int val) {
    node* newNode = new node(val);
    if(head == NULL) {
        head = newNode;
        return;
    }
    node* temp = head;
    while(temp->next != NULL) temp = temp->next;
    temp->next = newNode;
}
//function to get reference of the node to delete
node* getNode(node* head,int val) {
    while(head->num != val) head = head->next;

    return head;
}
//delete function as per the question
void deleteNode(node* t) {
    t->num = t->next->num;
    t->next = t->next->next;
    return;
}
//printing the list function
void printList(node* head) {
    while(head->next != NULL) {
        cout<<head->num<<"->";
        head = head->next;
    }
    cout<<head->num<<"\n";
}

int main() {
    node* head = NULL;
    //inserting node
    insertNode(head,1);
    insertNode(head,4);
    insertNode(head,2);
    insertNode(head,3);
    //printing given list
    cout<<"Given Linked List:\n";
    printList(head);
    node* t = getNode(head,2);
    //delete node
    deleteNode(t);
    //list after deletion operation
    cout<<"Linked List after deletion:\n";
    printList(head);
    return 0;
}
```

Output:

```
Given Linked List:
1->4->2->3
Linked List after deletion:
1->4->3
```

Time Complexity: O(1)

***Reason*:** It is a two-step process that can be obtained in constant time.

Space Complexity: O(1)

***Reason*:** No extra data structure is used.