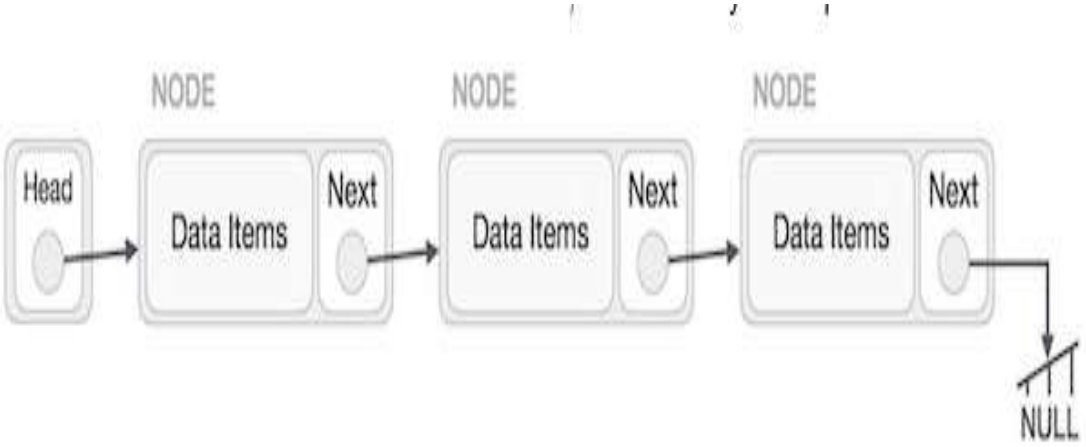
Unit 4. Linked List

- A linked list is a sequence of data structures, which are connected together via links.
- Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array. Following are the important terms to understand the concept of Linked List.
- **Link** − Each link of a linked list can store a data called an element.
- **Next** − Each link of a linked list contains a link to the next link called Next.
- **LinkedList** − A Linked List contains the connection link to the first link called First.

Linked List Representation

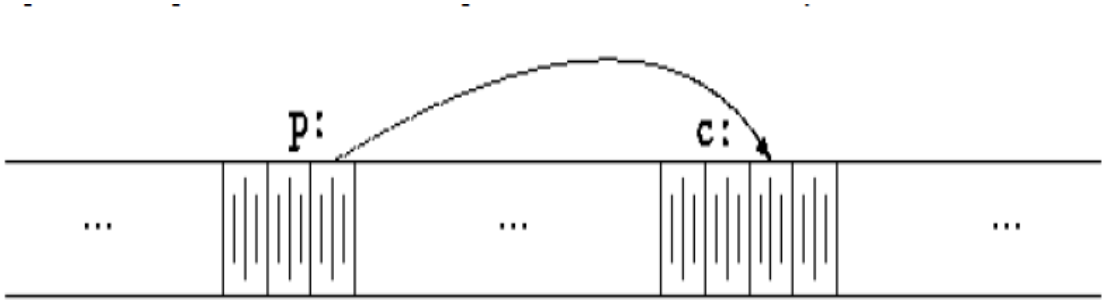
Linked list can be visualized as a chain of nodes, where every node points to the next node.



Pointers (**Not in your syllabus, this is just for your knowledge)

- A pointer is a variable that contains the address of a variable.
- A typical machine has an array of consecutively numbered or addressed memory cells that may be manipulated individually or in contiguous groups.
- One common situation is that any byte can be a char, a pair of one-byte cells can be treated as a short integer, and four adjacent bytes form a long. A pointer is a group of cells (often two or four) that can hold an address.

So if c is a char and p is a pointer that points to it, we could represent the situation this way:



- The unary operator & gives the address of an object, so the statement

  p = &c;

- assigns the address of c to the variable p, and p is said to ``point to" c.

- The & operator only applies to objects in memory: variables and array elements. It cannot be applied to expressions, constants, or register variables.
- The unary operator * is the *indirection or dereferencing operator; when applied to a pointer, it* accesses the object the pointer points to.
- Suppose that x and y are integers and ip is a pointer to int. This artificial sequence shows how to declare a pointer and how to use & and *:

```
int x = 1, y = 2, z[10];

int *ip; /* ip is a pointer to int */

ip = &x; /* ip now points to x */

y = *ip; /* y is now 1 */ *ip=0 &x=0

*ip = 0; /* x is now 0 */

ip = &z[0]; /* ip now points to z[0] */
```

- If ip points to the integer x, then *ip can occur in any context where x could, so

```
*ip = *ip + 10;        increments *ip by 10.
```

- The unary operators * and & bind more tightly than arithmetic operators, so the assignment

```
y = *ip + 1
```

takes whatever ip points at, adds 1, and assigns the result to y, while

*ip += 1 increments what ip points to, as do

++*ip and (*ip)++

- The parentheses are necessary in this last example; without them, the expression would increment ip instead of what it points to, because unary operators like * and ++ associate right to left.

Call By value

- swap(a, b);

  where the swap function is defined as

```
void swap(int x, int y) /* WRONG */
{
    int temp;
    temp = x;
    x = y;
    y = temp;
}
```

- Because of call by value, swap can't affect the arguments a and b in the routine that called it
- The way to obtain the desired effect is for the calling program to pass *pointers to the values to* be changed:

```
swap(&a, &b);
```

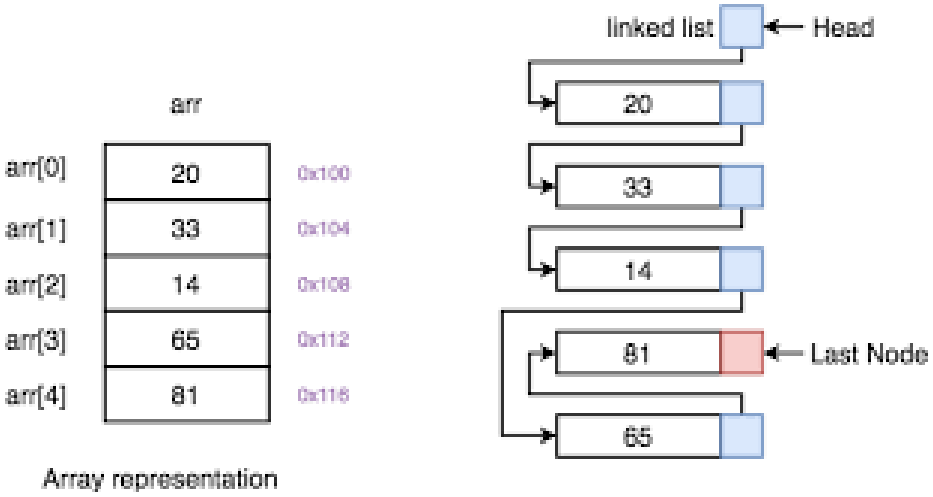
- Since the operator & produces the address of a variable, &a is a pointer to a. In swap itself, the parameters are declared as pointers, and the operands are accessed indirectly through them.

```
void swap(int *px, int *py) /* interchange *px and *py */
{
    int temp;
    temp = *px;
    *px = *py;
    *py = temp;
}
```

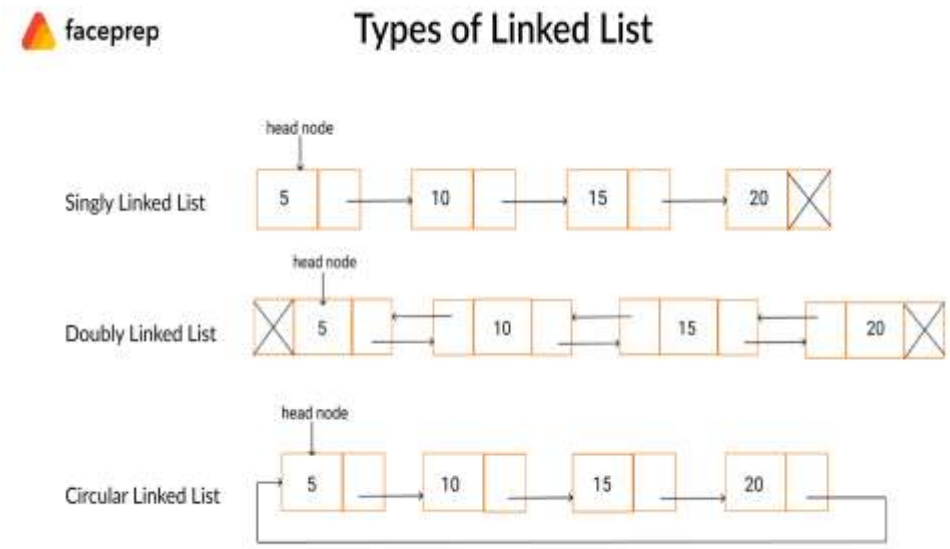
Types of Linked List

- Simple Linked List − Item navigation is forward only singly
- Doubly Linked List − Items can be navigated forward and backward.
- Circular Linked List − Last item contains link of the first element as next and the first element has a link to the last element as previous.

Array verses Linked list



linked list diagrammatical representation



Insertion Operation for Singly Linked List

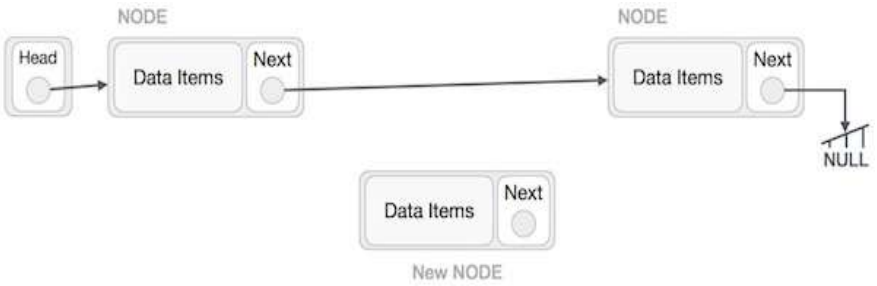
- **Steps to insert node at the middle of singly linked list**

Algorithm to insert node at the beginning of Singly Linked List

1. Create a new node
2. Traverse to the `n-1`th position of the linked list and connect the new node with the `n+1`th node. Means the new node should also point to the same node that

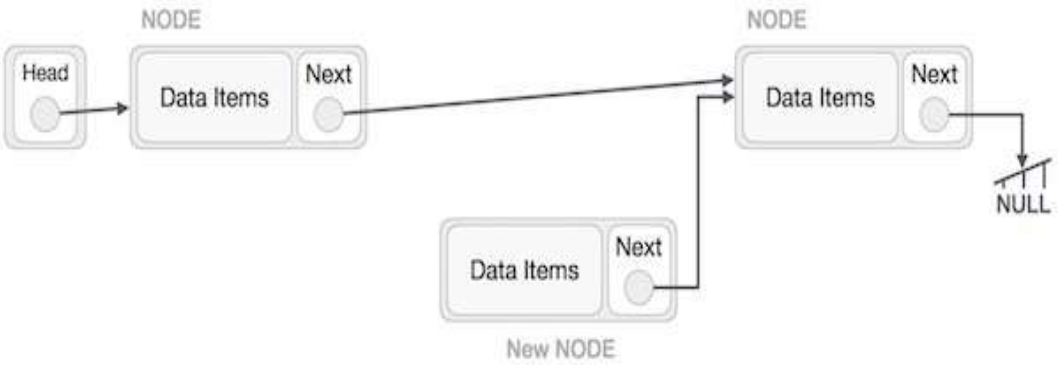
the `n-1th` node is pointing to. (`newNode->next = temp->next` where `temp` is the `n-1th` node).

Step1 : First, create a node using the same structure and find the location where it has to be inserted.



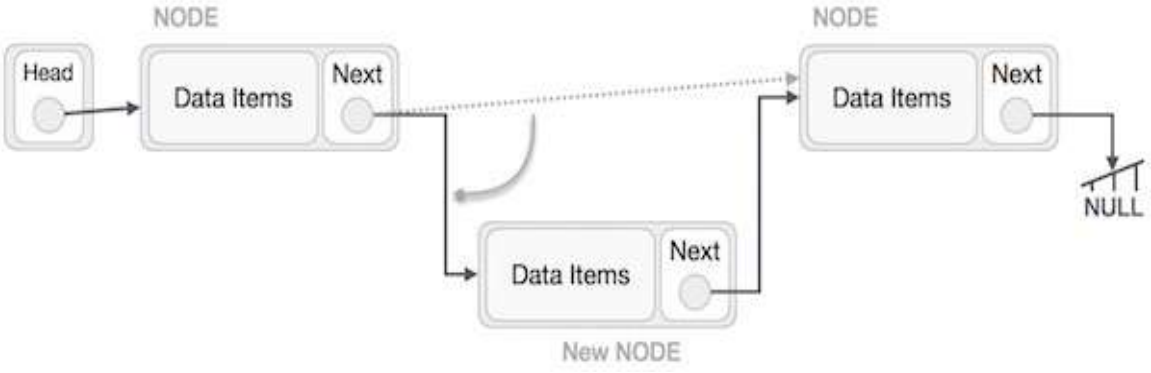
Step2: Imagine that we are inserting a node **B** (NewNode), between **A** (LeftNode)and **C** (RightNode).Then point B next to C

NewNode.next −> RightNode (c);

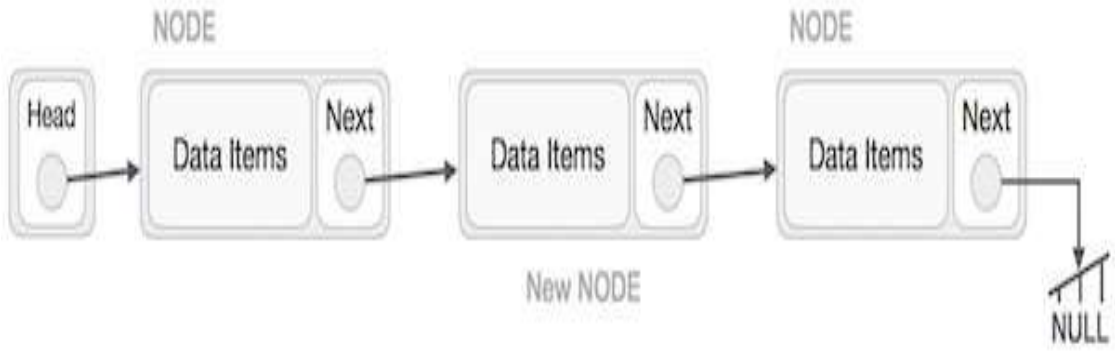


Step3: Now, the next node at the left should point to the new node.

LeftNode.next(a.next) −> NewNode (b);



Step 4: This will put the new node in the middle of the two. The new list should look like this –

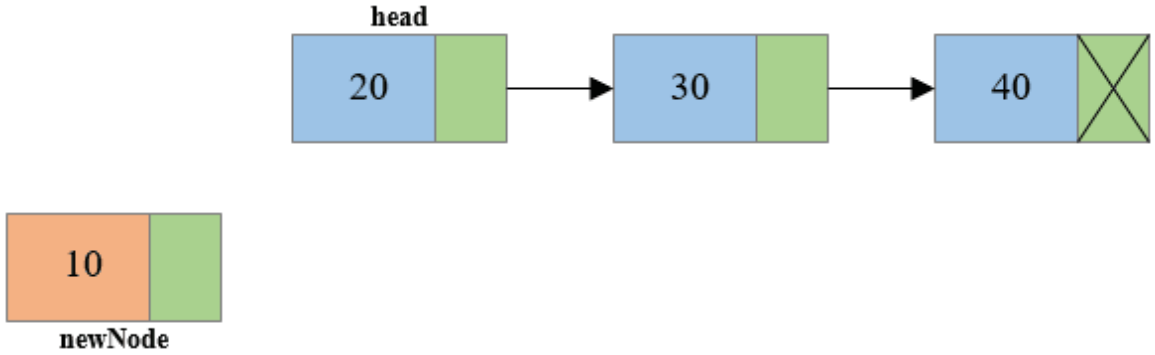


- **Steps to insert node at the beginning of singly linked list**

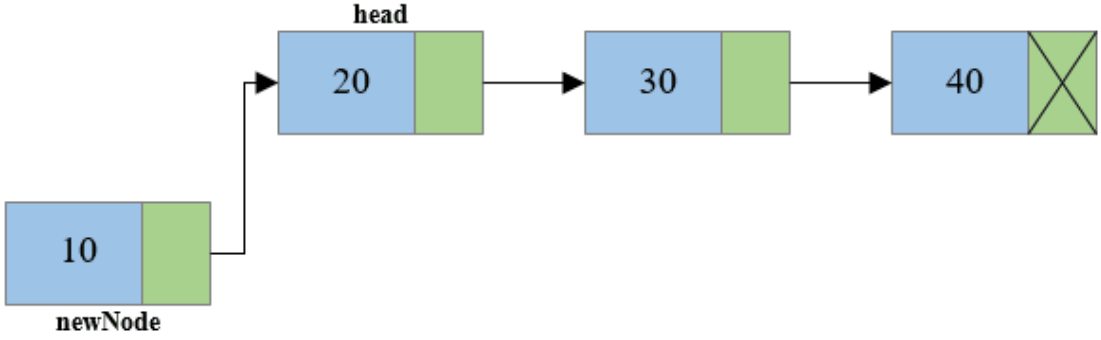
Algorithm to insert node at the beginning of Singly Linked List

1. Declare a head pointer and make it as NULL.
2. Create a new node with the given data.
3. Make the new node points to the head node.
4. Finally, make the new node as the head node.

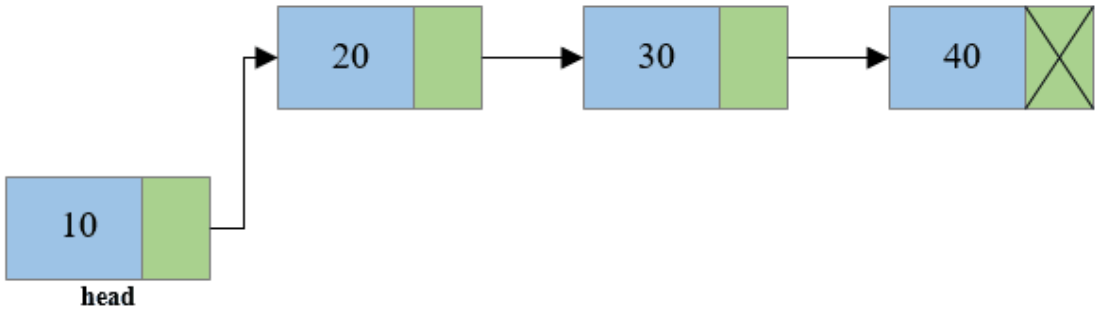
1. Create a new node, say newNode points to the newly created node.



2. Link the newly created node with the head node, i.e. the newNode will now point to head node.



3. Make the new node as the head node, i.e. now head node will point to newNode.



- **Steps to insert node at the end of singly linked list**

Algorithm to insert node at the end of Singly Linked List

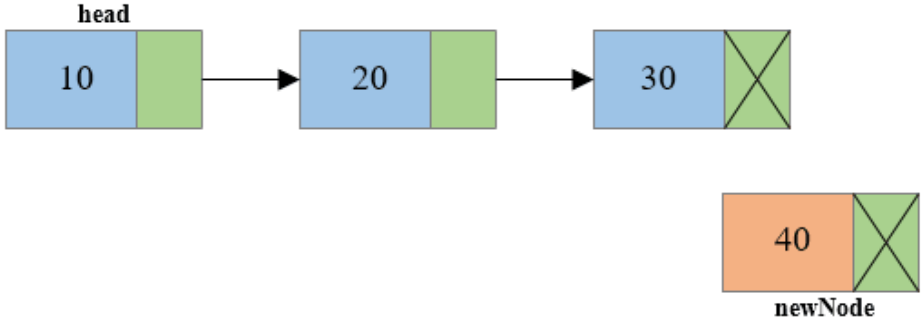
1. Declare head pointer and make it as NULL.

2. Create a new node with the given data. And make the new node => next as NULL.
   (Because the new node is going to be the last node.)

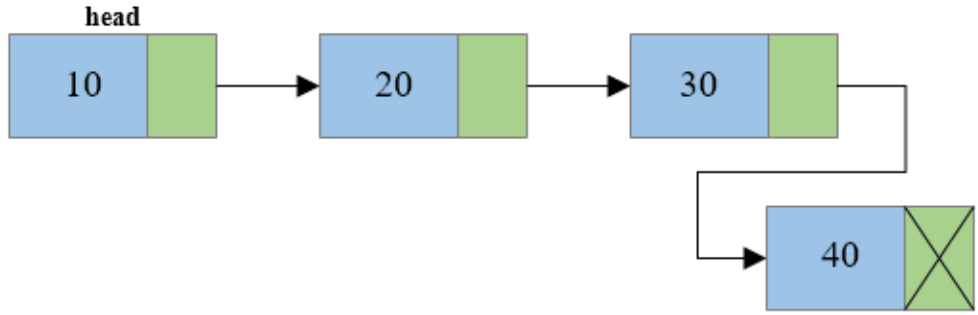
3. If the head node is NULL (Empty Linked List),
   make the new node as the head.

4. If the head node is not null, (Linked list already has some elements),
   find the last node.
   make the last node => next as the new node.

1. Create a new node and make sure that the address part of the new node points to NULL i.e. newNode->next=NULL



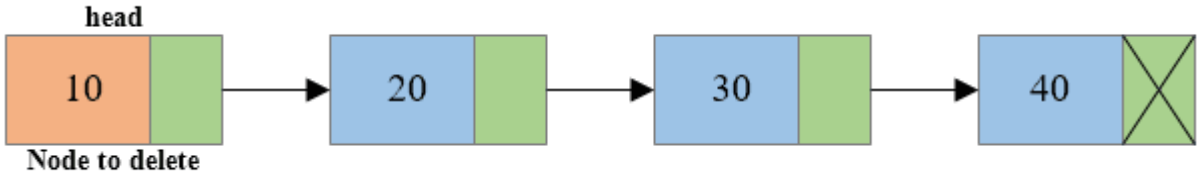
2. Traverse to the last node of the linked list and connect the last node of the list with the new node, i.e. last node will now point to new node. (lastNode->next = newNode).



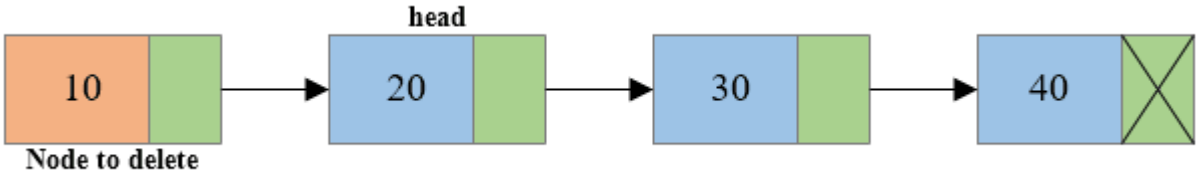
Deletion Operation

- **Steps to delete first node of a Singly Linked List**

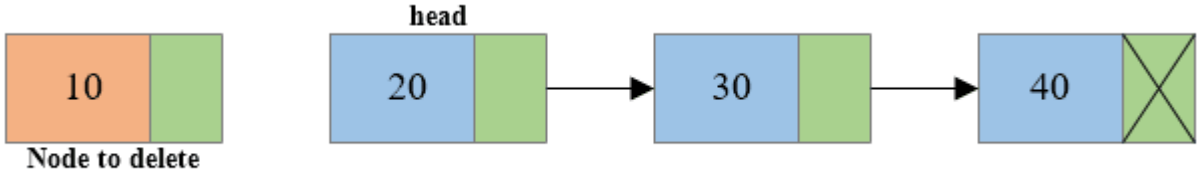
1. Copy the address of first node i.e. head node to some temp variable say toDelete.



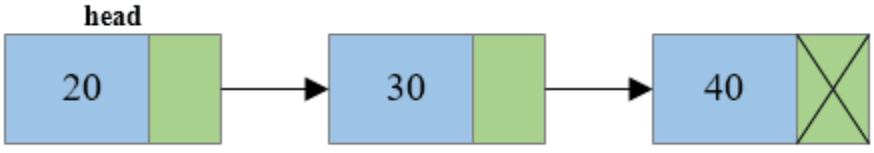
2. Move the head to the second node of the linked list i.e. head = head -> next.



3. Disconnect the connection of first node to second node.

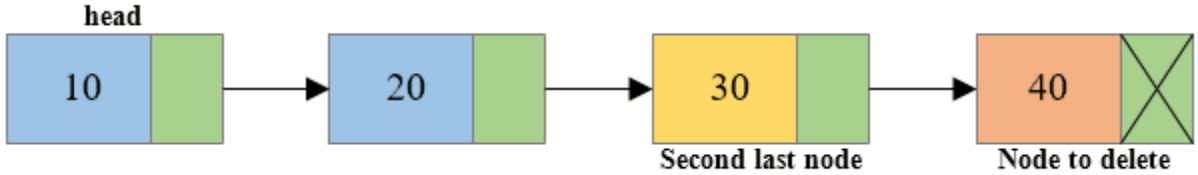


4. Free the memory occupied by the first node.

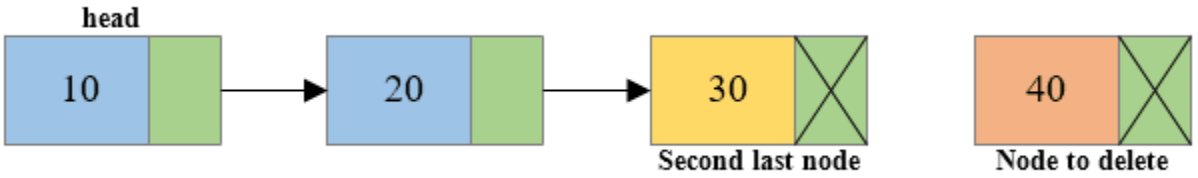


- **Steps to delete last node of a Singly Linked List**

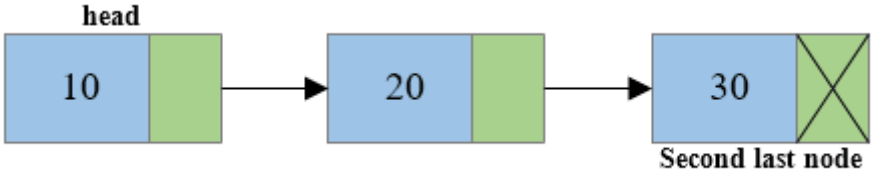
1. Traverse to the last node of the linked list keeping track of the second last node in some temp variable say secondLastNode.



2. If the last node is the head node then make the head node as NULL else disconnect the second last node with the last node i.e. secondLastNode->next = NULL.

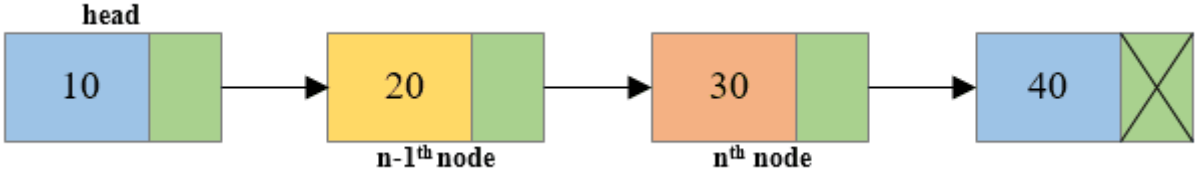


3. Free the memory occupied by the last node.



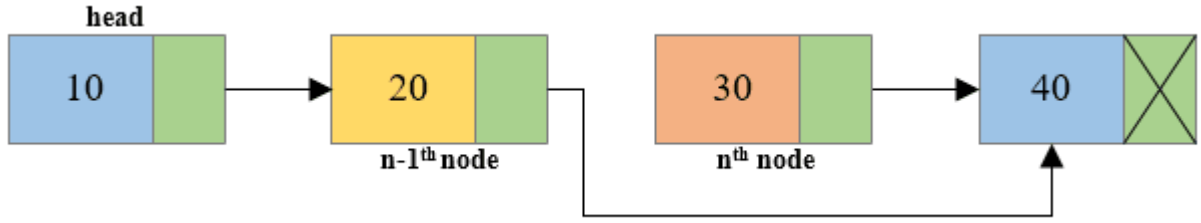
- **Steps to delete middle node of Singly Linked List**

1. Traverse to the n^{th} node of the singly linked list and also keep reference of $n-1^{th}$ node in some temp variable say prevnode.

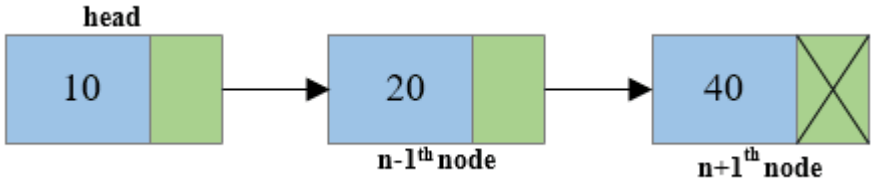


2. Reconnect the $n-1^{th}$ node with the $n+1^{th}$ node i.e. prevNode->next = toDelete-> next (Where prevNode is $n-1^{th}$ node and toDelete node is the n^{th} node

and toDelete->next is the n+1th node).



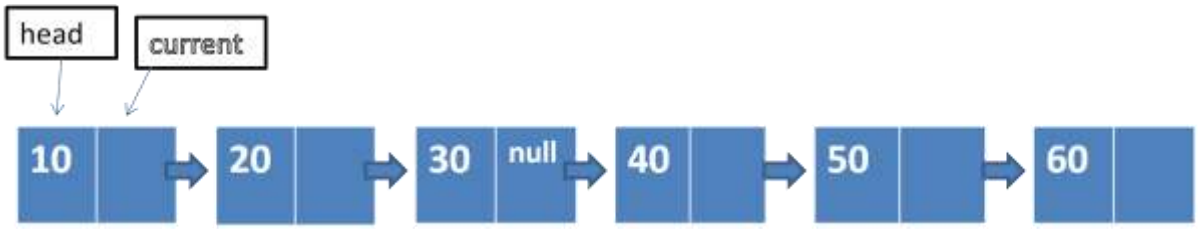
3. Free the memory occupied by the nth node i.e. toDelete node.



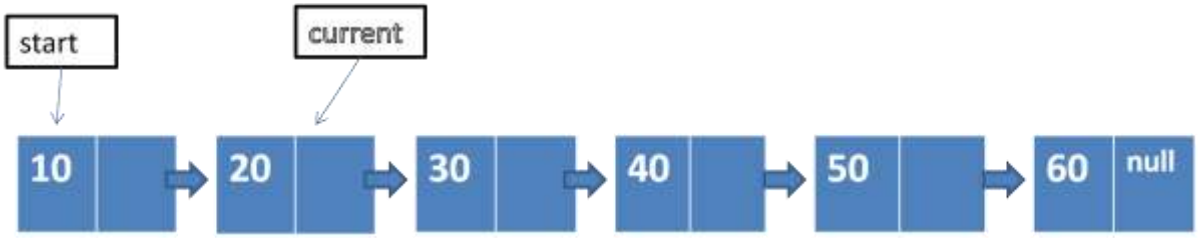
Searching a value in the linked list

step 1 : assigning current as the first node
step 2: begin traversing the linked list
while(current!=NULL and current->data!= 30 val):
current = current -> next

current =30



In this case current data is not equal to 30, hence move to second node.
current = current -> next



In this case current data is not equal to 30, hence move to third node.
current = current -> next
now current ->data= 30 so the value is found