Linked List

Deleting a node

Operations on single linked list

- Traversing a list
    - Printing, finding minimum, etc.
- Insertion of a node into a list
    - At front, end and anywhere, etc.
- Deletion of a node from a list
    - At front, end and anywhere, etc.
- Comparing two linked lists
    - Similarity, intersection, etc.
- Merging two linked lists into a larger list
    - Union, concatenation, etc.
- Ordering a list
    - Reversing, sorting, etc.

Deleting a node

The node currently pointed to by the pointer is deallocated, and the pointer is considered unassigned. The memory is returned to the free store.

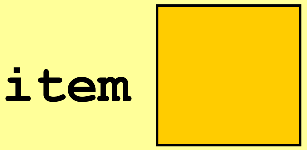
When Deleting….

- The list should not be empty.
  - The value associated with the start in a pointer. If the pointer value is **NULL**, it means that the list is empty. i.e it is the case of **UNDERFLOW** of the list.

When Deleting the node existing in the list……

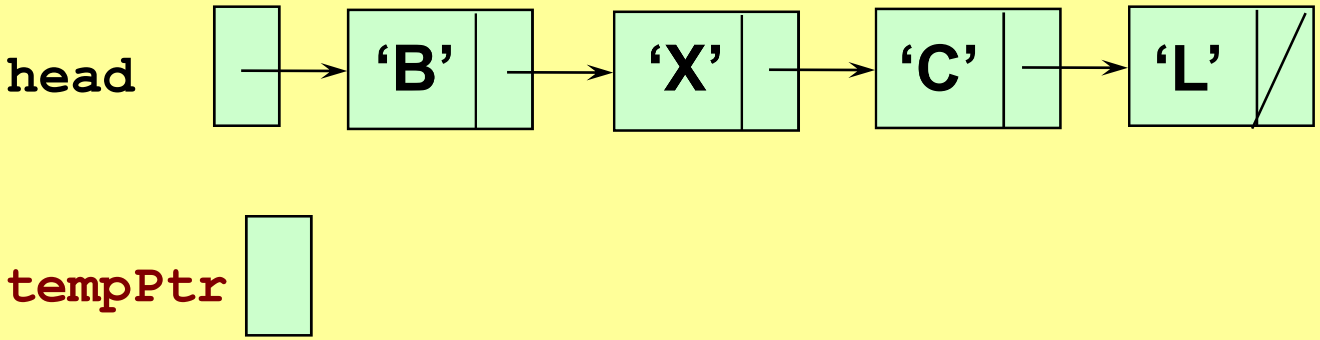
- The list should not be empty.
  - Store the value associated with the start in a pointer. If the pointer value is **NULL**, it means that the list is empty. i.e it is the case of **UNDERFLOW** of the list.

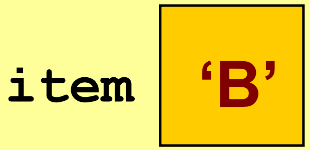
CASE 1: Deleting the first node from the list



```
NodeType *tempPtr;

item = head->info;
tempPtr = head;
head = head->next;
free(tempPtr);
```

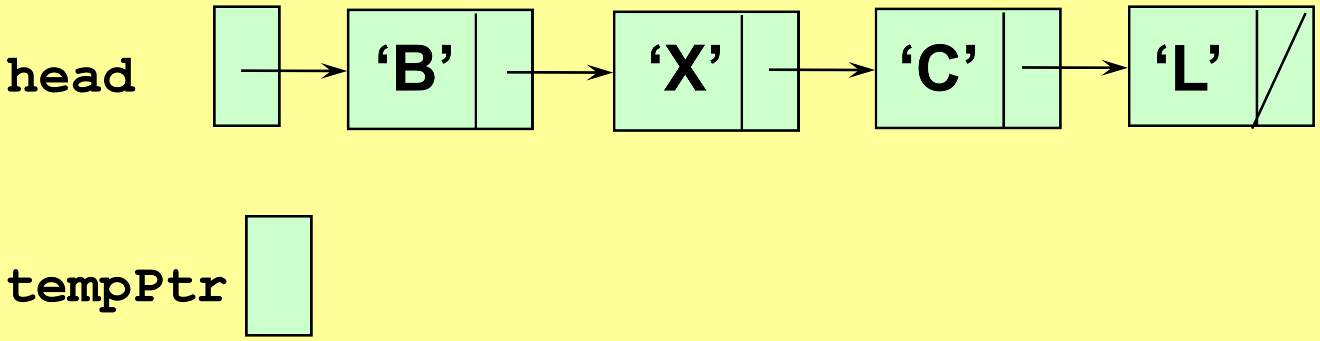




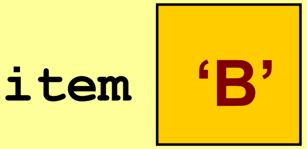
Deleting the first node from the list

```
    NodeType *tempPtr;

    item = head->info;
    tempPtr = head;
    head = head->next;
    free(tempPtr);
```

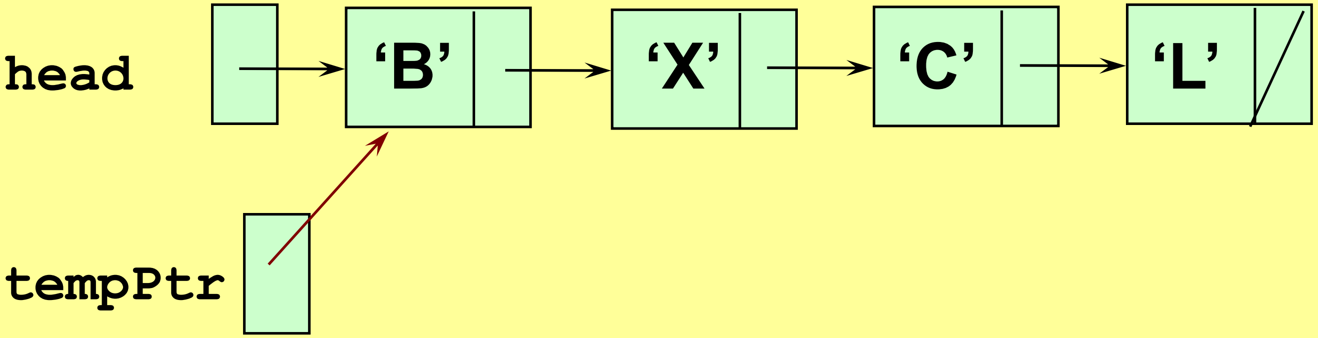


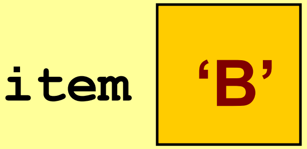
Deleting the first node from the list



```
NodeType *tempPtr;

item = head->info;
tempPtr = head;
head = head->next;
free(tempPtr);
```

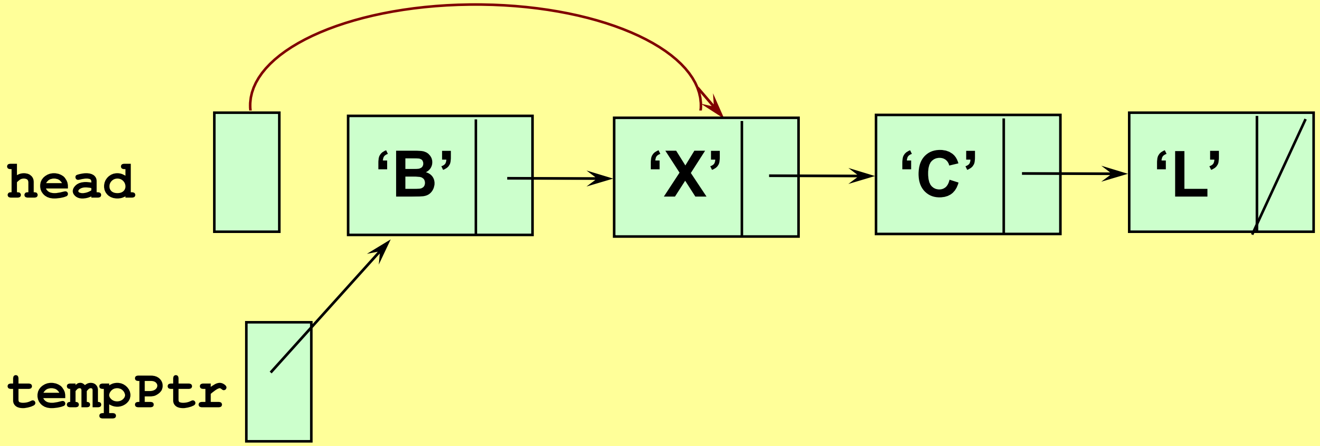


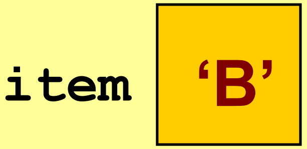


Deleting the first node from the list

```
NodeType *tempPtr;

item = head->info;
tempPtr = head;
head = head->next;
free(tempPtr);
```

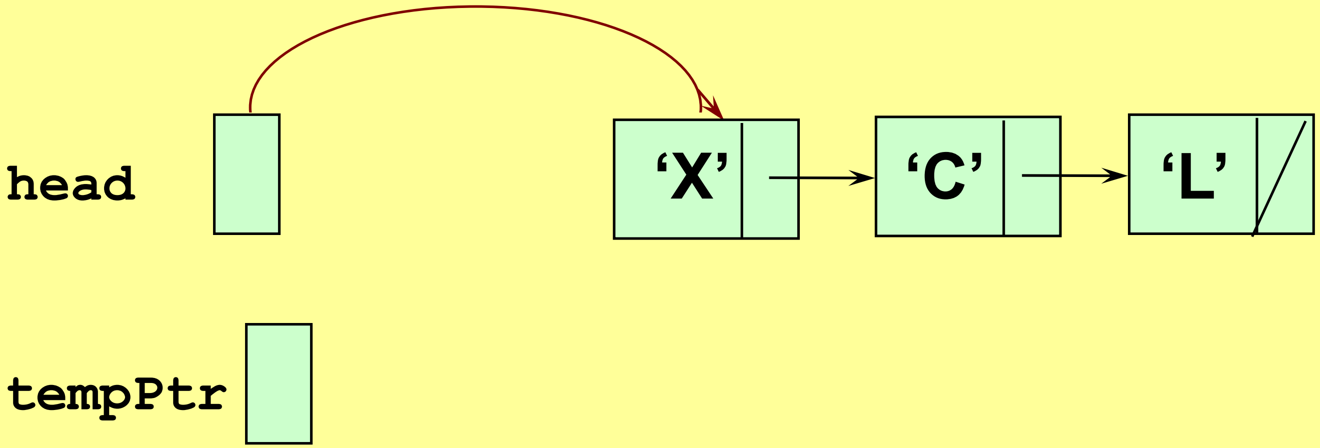




Deleting the first node from the list

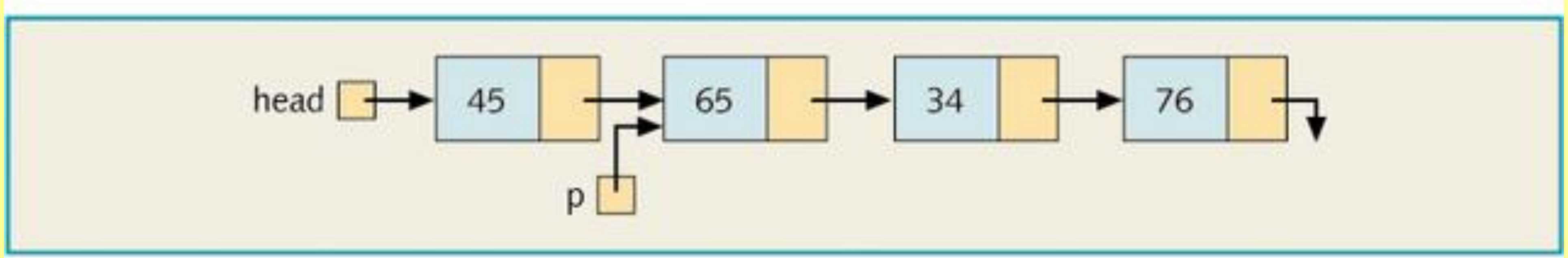
```
NodeType *tempPtr;

item = head->info;
tempPtr = head;
head = head->next;
free(tempPtr);
```



CASE 2(a): Deleting the node with a specific value from the list

- Consider the following linked list:

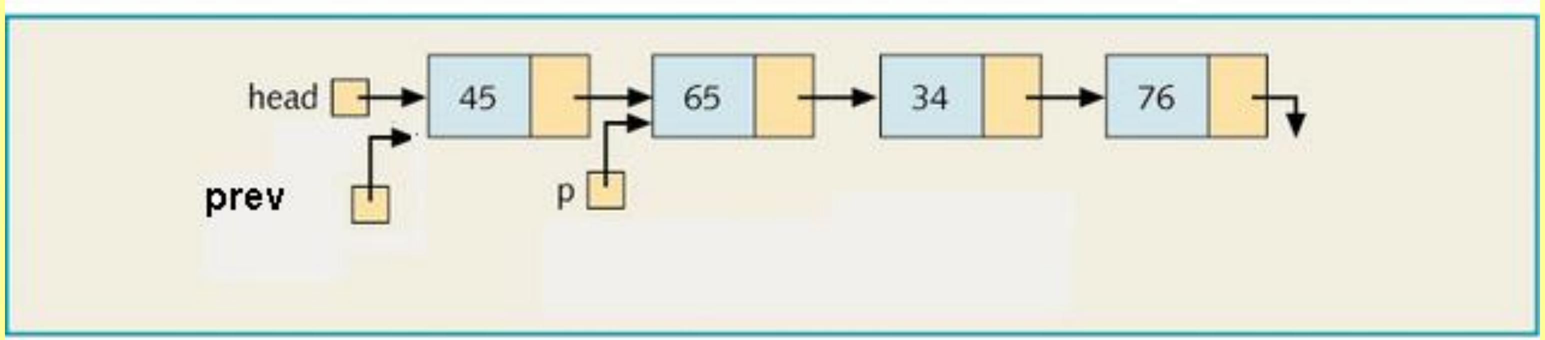


Linked list before item **deleted**

- Example: A node with info 65 <u>is to be deleted</u> pointed by p

CASE 2(b): Deleting a specific node (number) from the list

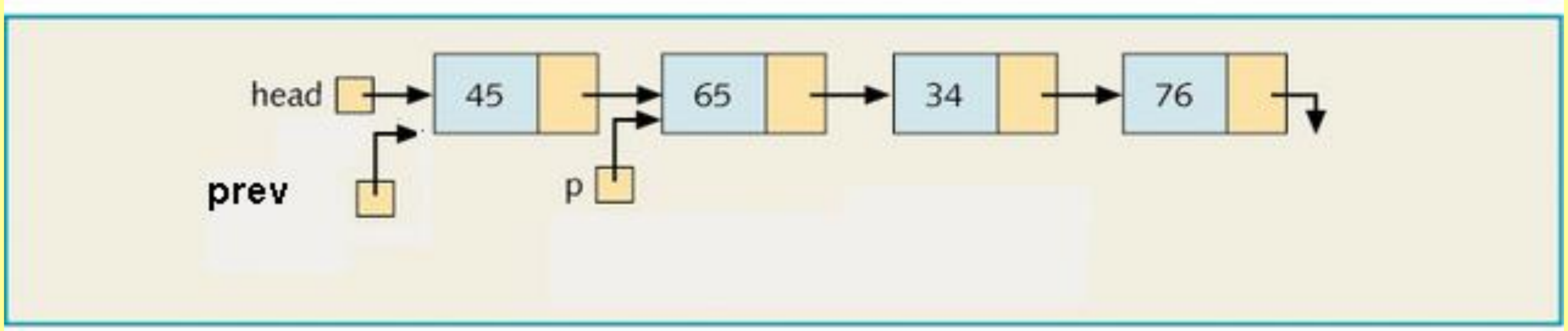
```
Node_number = 1;
struct node *p=head ;
struct node *prev;
        While(p!=NULL) //loop to locate node
        { if (node_number) != delete_node)
                  {
                        prev = p;  //prev is a pointer pointing to node previous to P
                        p = p-> next;
                        node_number ++;
                  }
            else break;
        }
```



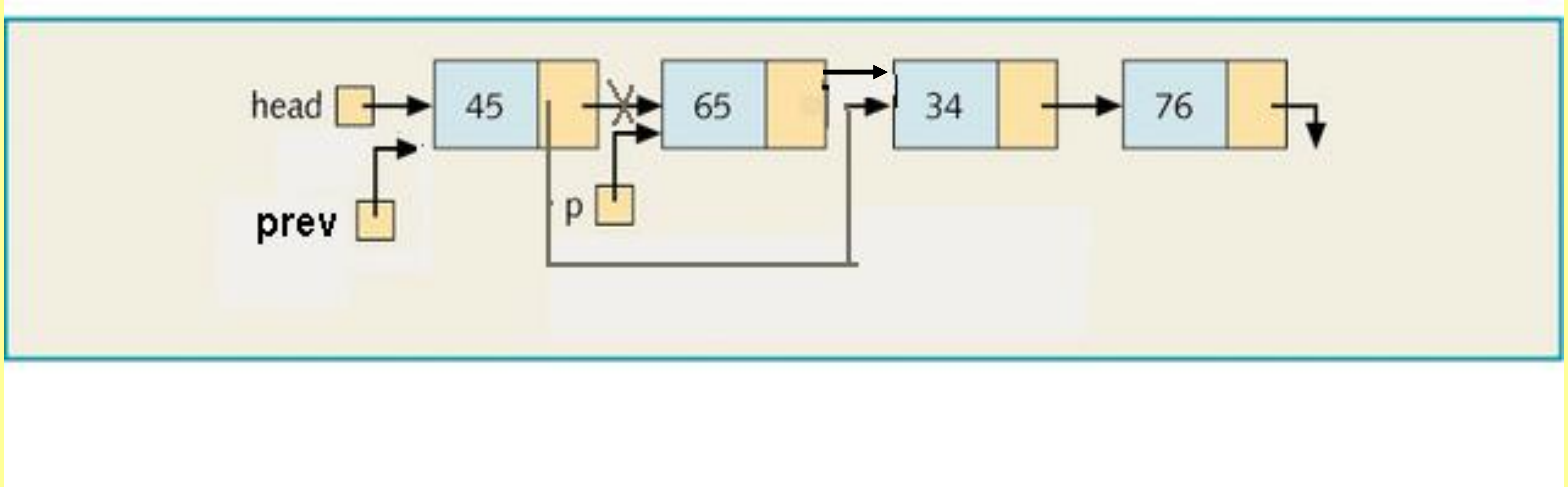
item 65

```
Item = p->info;
prev->next =p->next;
free(p);
```

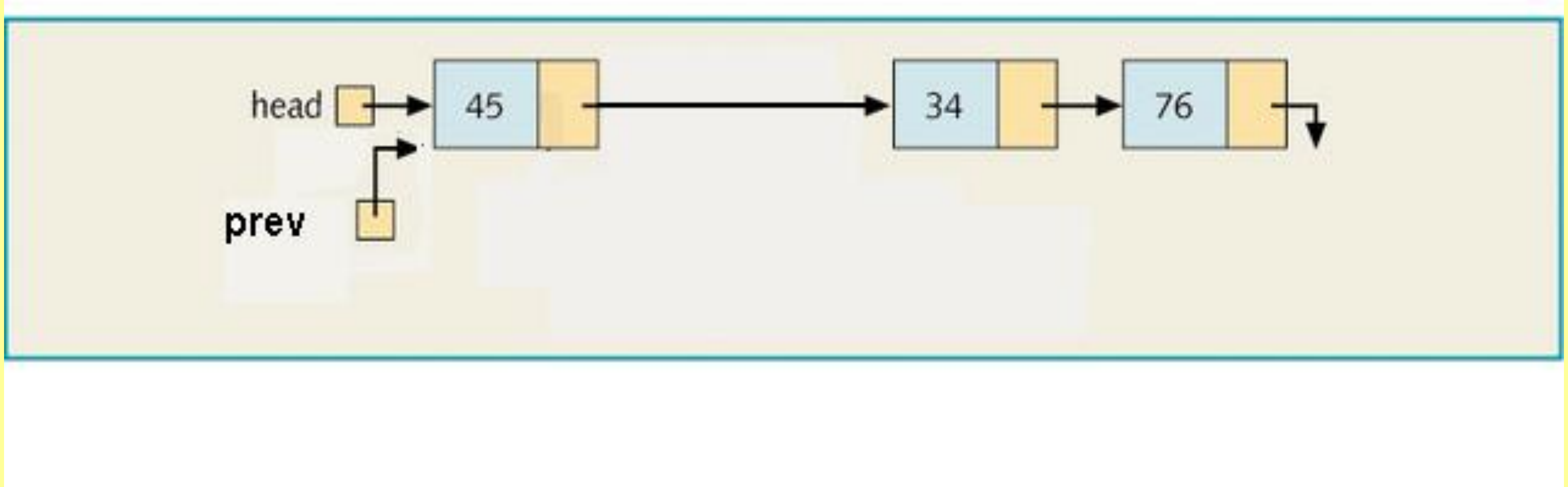
Item = p->info;



prev->next = p->next;



Free(p);



Deleting last node

Steps required for deleting the node:-

- If the Linked list has only one node then make head node null
- Else traverse to the end of the linked list
- While traversing store the previous node i.e. 2nd last node
- Change the next of 2nd last node to null
- Free/delete memory of the the last node
- Now, 2nd last node becomes the last node.

Deleting last node

Steps required for deleting the node:-

- If the Linked list has only one node then –
  - make head node null
- Else traverse to the end of the linked list
- While traversing store the previous node i.e. 2nd last node
- Change the next of 2nd last node to null
- Free/delete memory of the last node
- Now, 2nd last node becomes the last node.

- Write the function DelLastNode() to delete the last node of the list.

Operations on Linked Lists

Operations on single linked list

- Traversing a list
  - Printing, finding minimum, etc.
- Insertion of a node into a list
  - At front, end and anywhere, etc.
- Deletion of a node from a list
  - At front, end and anywhere, etc.
- Comparing two linked lists
  - Similarity, intersection, etc.
- Merging two linked lists into a larger list
  - Union, concatenation, etc.
- Ordering a list
  - Reversing, sorting, etc.

Search an element in a Linked List

- Follow the below steps to solve the problem:
  - Initialize a node pointer, current = head.
  - Do following while current is not NULL
    - If the current value (i.e., current->key) is equal to the key being searched return true.
    - Otherwise, move to the next node (current = current->next).
  - If the key is not found, return false

Write a function to count the number of nodes in a given singly linked list

- Follow the given steps to solve the problem:
  - Initialize count as 0
  - Initialize a node pointer, current = head.
  - Do following while current is not NULL
    - current = current -> next
    - Increment count by 1.
  - Return count

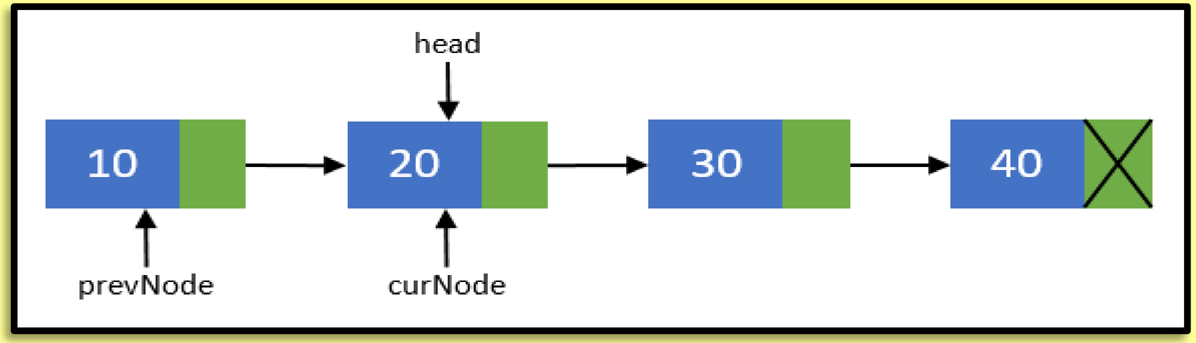
Reverse a Linked List

Single Linked List: Reversing

Steps to reverse a Singly Linked List using Iterative method

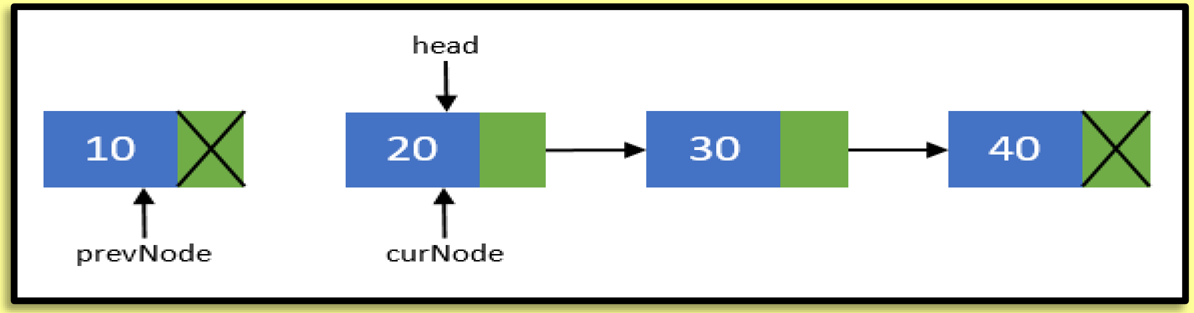
Step 1: Create two more pointers other than **head** namely **prevNode** and **curNode** that will hold the reference of previous node and current node respectively.

- Make sure that prevNode points to first node i.e. `prevNode = head.`
- head should now point to its next node i.e. `head = head->next.`
- curNode should also points to the second node i.e. `curNode = head.`

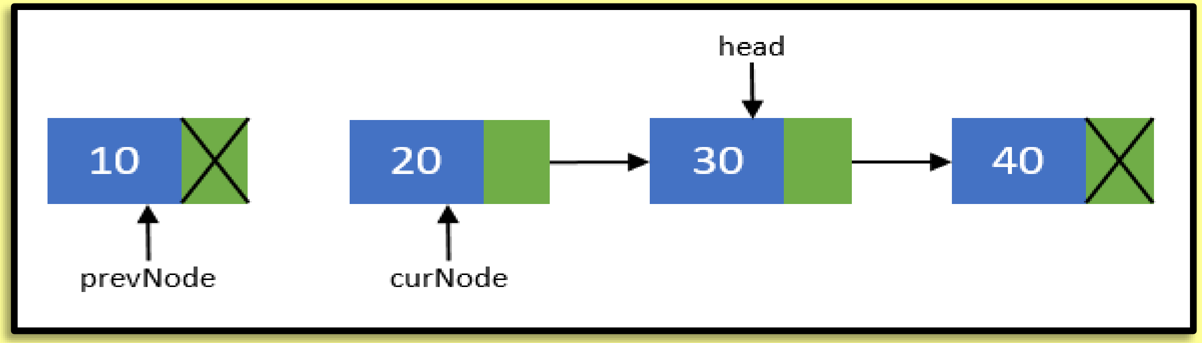


Reversing a List

Step 2: Now, disconnect the first node from others. We will make sure that it points to none. As this node is going to be our last node. Perform operation `prevNode->next = NULL.`

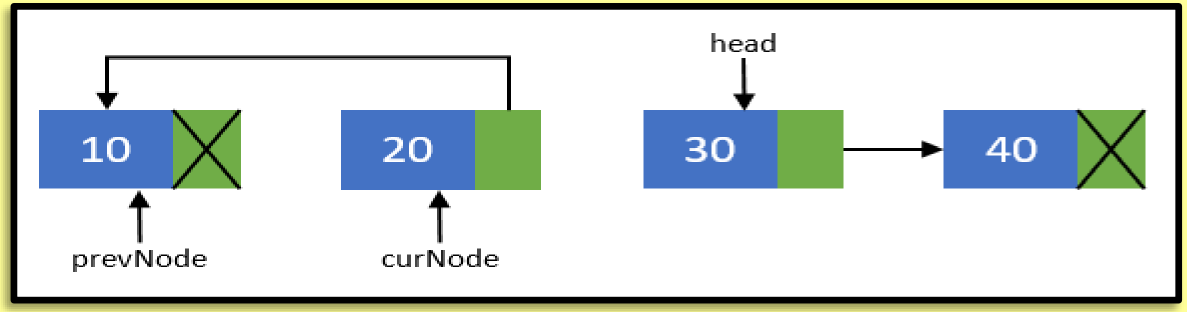


Step 3: Move the head node to its next node i.e. `head = head->next.`

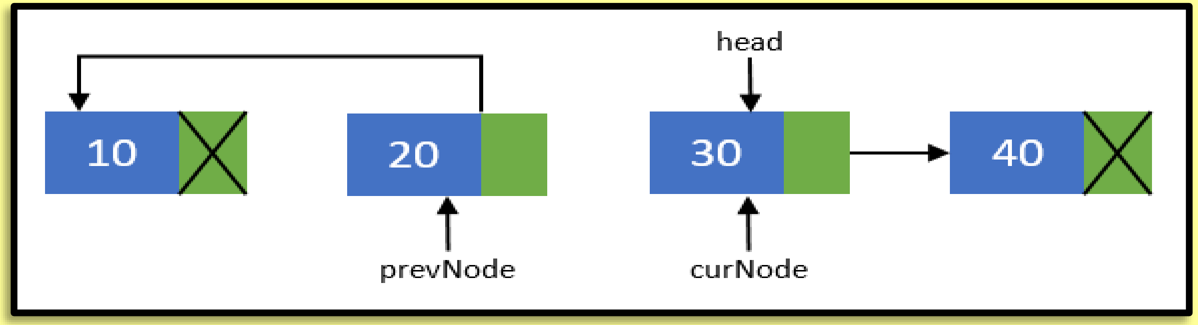


Reversing a List

Step 4: Now, re-connect the current node to its previous node i.e. `curNode->next = prevNode;`



Step 5: Point the previous node to current node and current node to head node. Means they should now point to `prevNode = curNode;` and `curNode = head.`

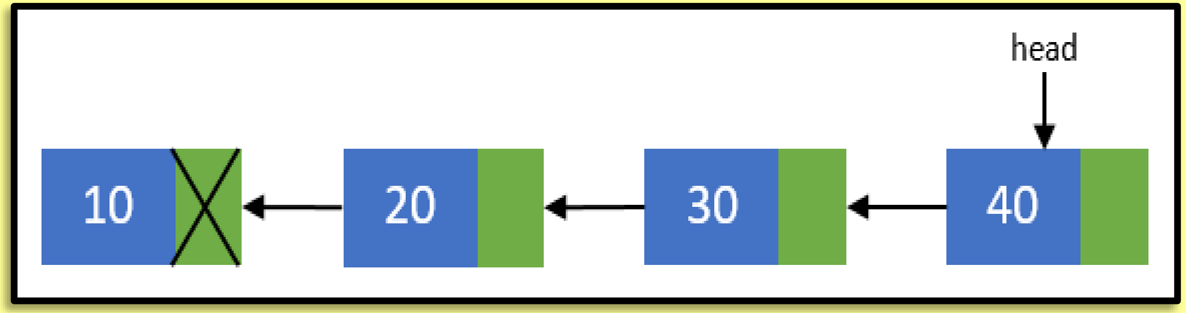


Reversing a List

Step 6: **Repeat** steps 3-5 till head pointer becomes NULL.

Step 7: Now, after all nodes has been re-connected in the reverse order. Make the last node as the first node. Means the head pointer should point to prevNode pointer.

- Perform **`head = prevNode;`** And finally you end up with a reversed linked list of its original.



- Write function RevList() to reverse the list.

Sort List using Bubble Sort

```
/* Bubble sort the given linked list */
void bubbleSort( )
{
    int swapped, i;
    struct Node *ptr1;
    struct Node *lptr = NULL;

    /* Checking for empty list */
    if (start == NULL)
        return;
```

Bubble Sort

```
 do
    {
        swapped = 0;
        ptr1 = start;

        while (ptr1->next != lptr)
        {
            if (ptr1->data > ptr1->next->data)
            {
                swap(ptr1, ptr1->next);
                swapped = 1;
            }
            ptr1 = ptr1->next;
        }
        lptr = ptr1;
    }
    while (swapped);
}
```

```
// function to swap data of two nodes a and b
void swap(struct Node *a, struct Node *b)
{
    int temp = a->data;
    a->data = b->data;
    b->data = temp;
}
```