# Insertion in single linked list

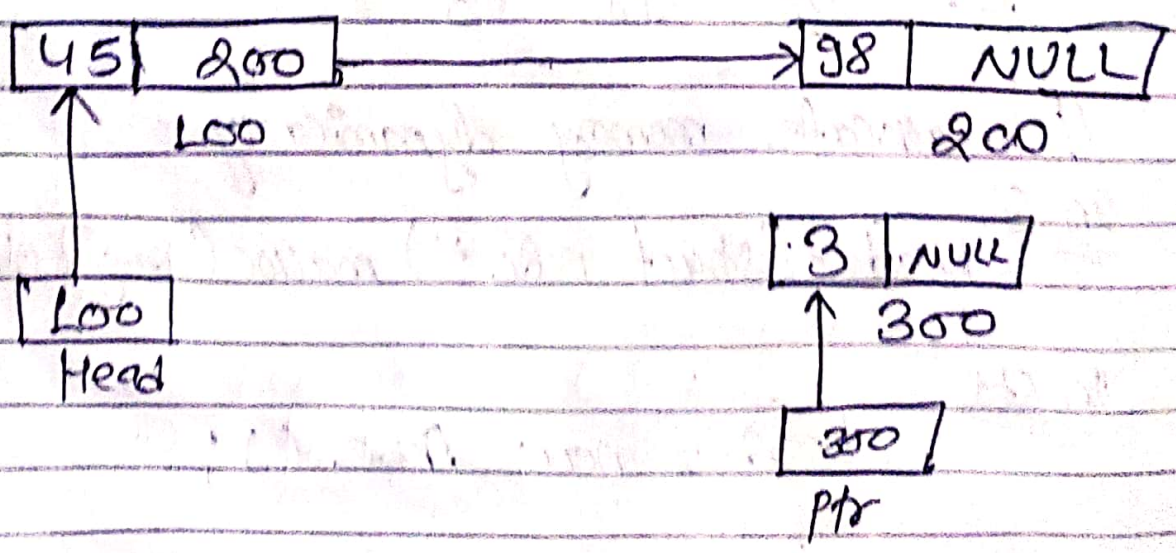
(I) Insert at the beginning of linked list.
↳ Complexity → O(1).

(II) Insert between two node.
↳ Complexity → O(n).

(III) Insert at the end of linked list.
↳ Complexity → O(n).

(IV) Insert after a node
↳ Complexity → O(1).

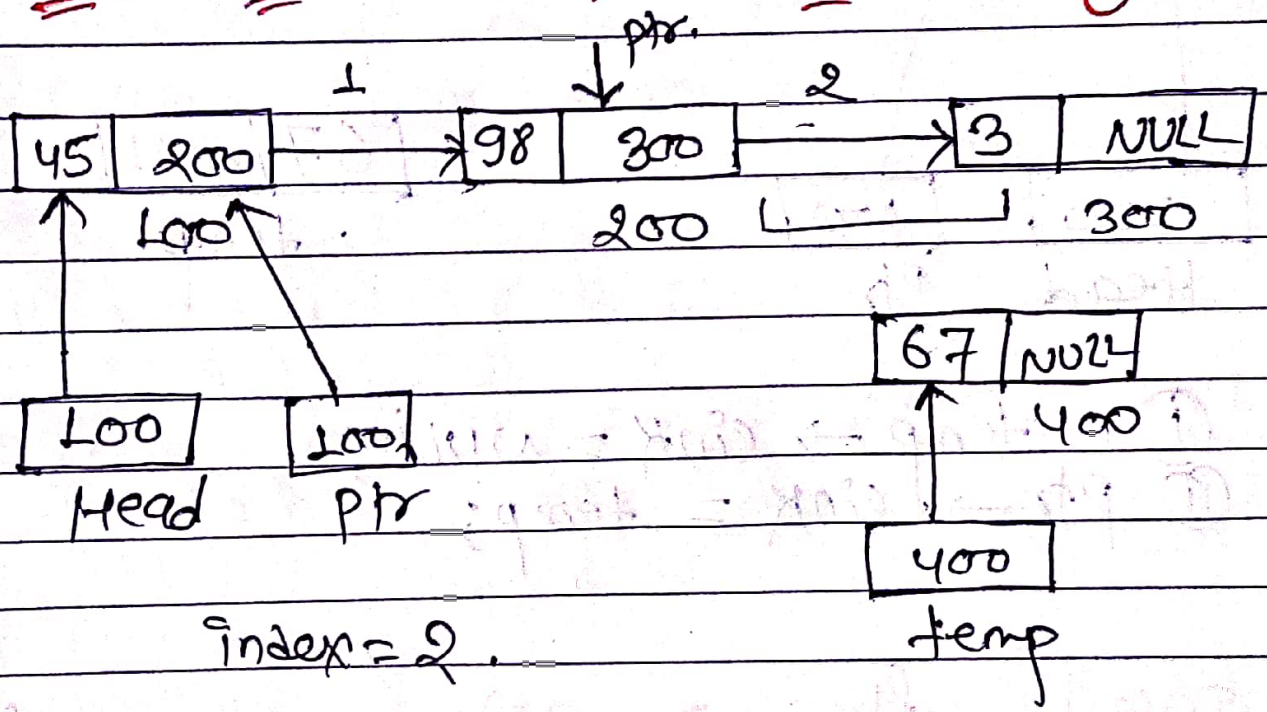
## Insert at the beginning O(1)



(i) ptr → link = Head;
(ii) Head = ptr;

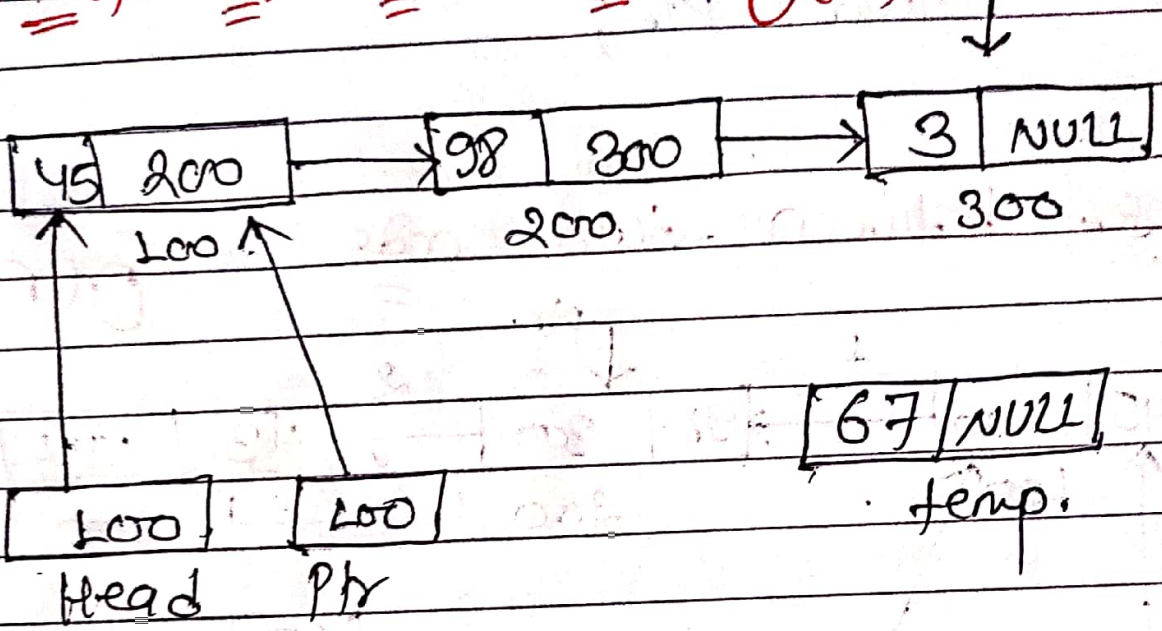
## Insert between two node

O(n)



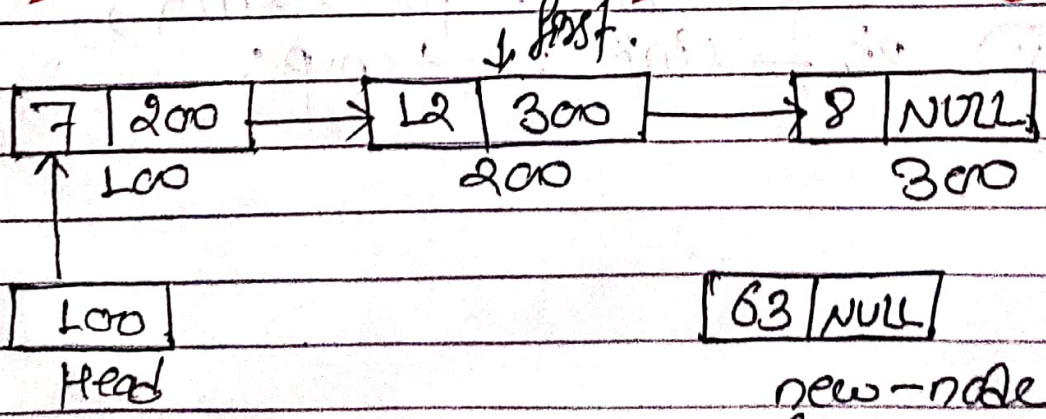

(i) temp → link = ptr → link;
(ii) ptr → link = temp;

## Insert at the end = O(n)




(I) temp → link = NULL;
(II) ptr → link = temp;

## Insert after a node = O(1)

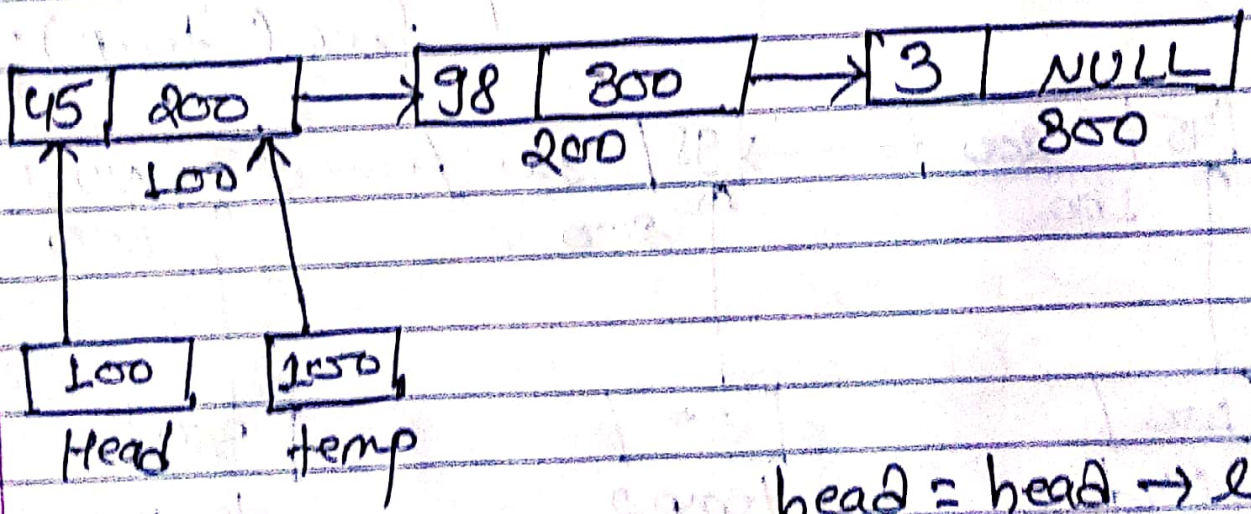



(I) new-node → link = first → link;
(II) first → link = new-node;

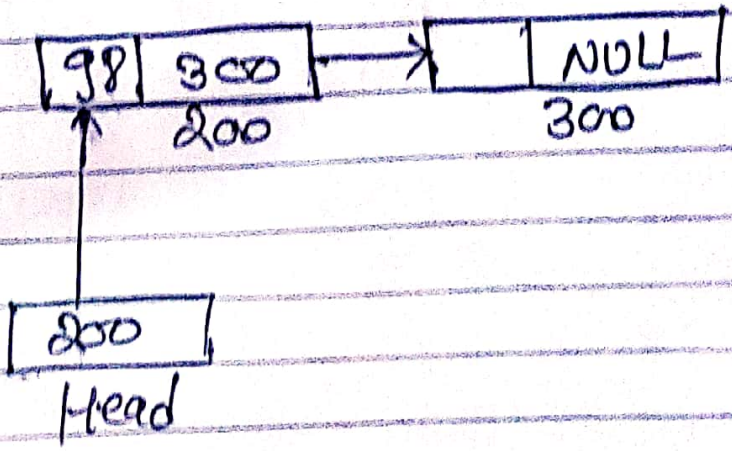
# Deletion in Single linked list

(i) Deleting first node
(ii) Deleting last node
(iii) Deleting a node in between
(iv) Deleting a node with given value.
(v) Deleting entire linked list.

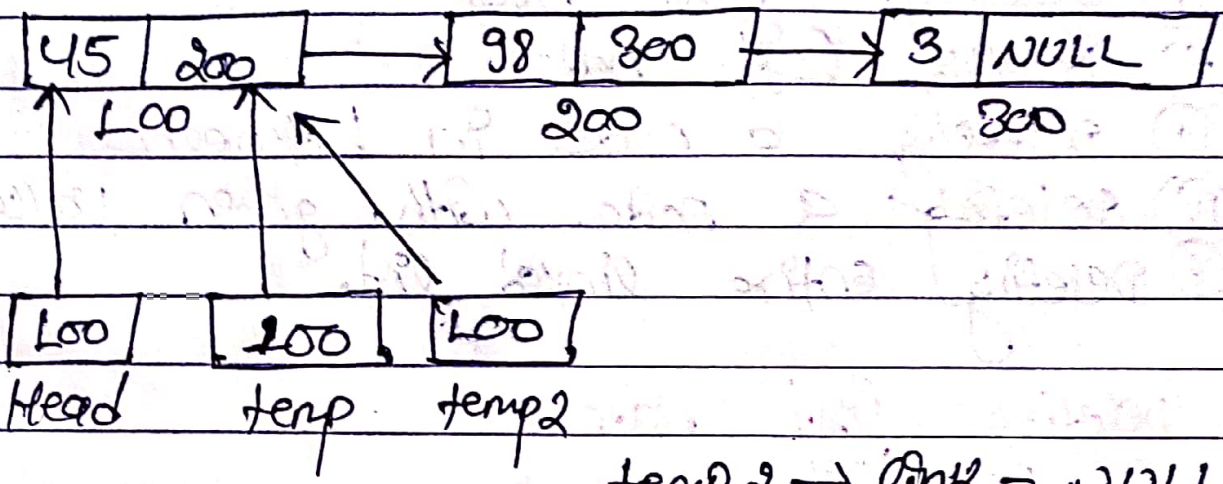
## Deleting first node




```
head = head -> link;
free (temp);
```

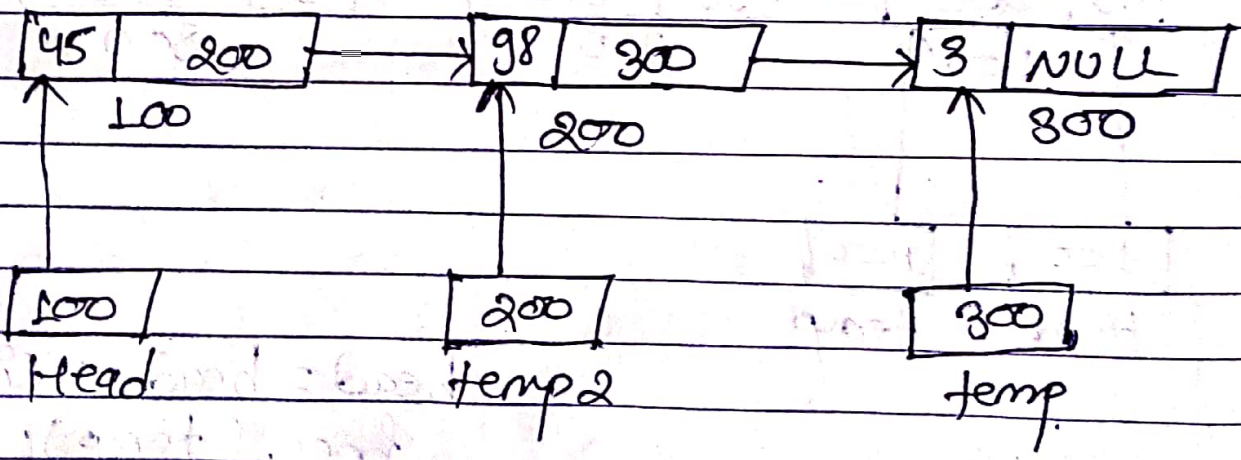


# Deleting last node

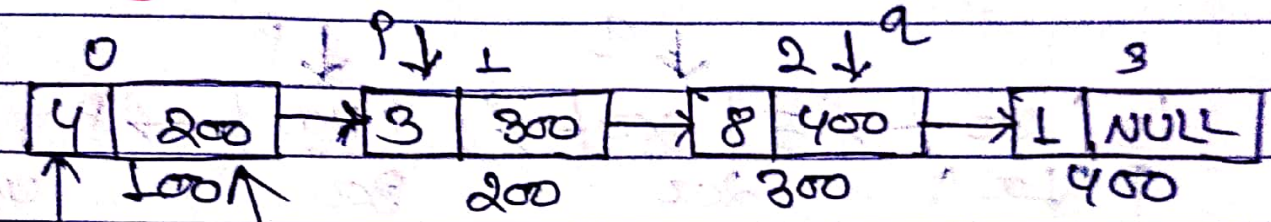



temp2 → link = NULL

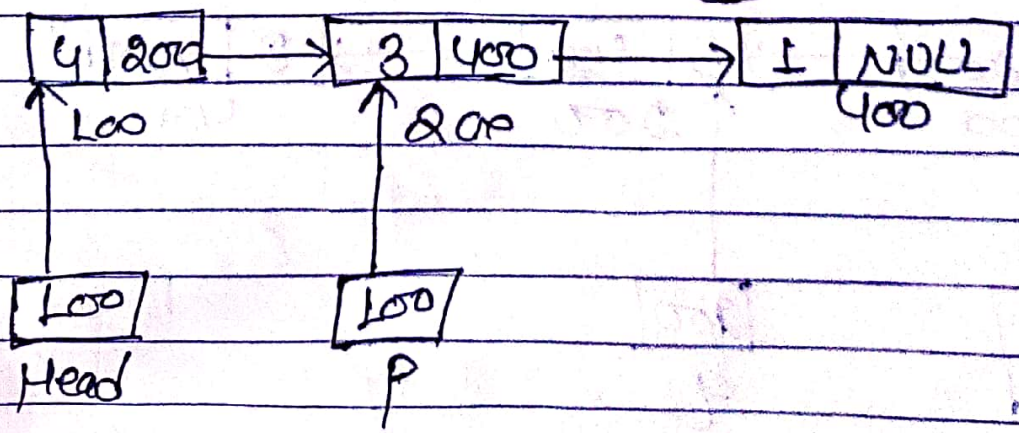
free(temp);



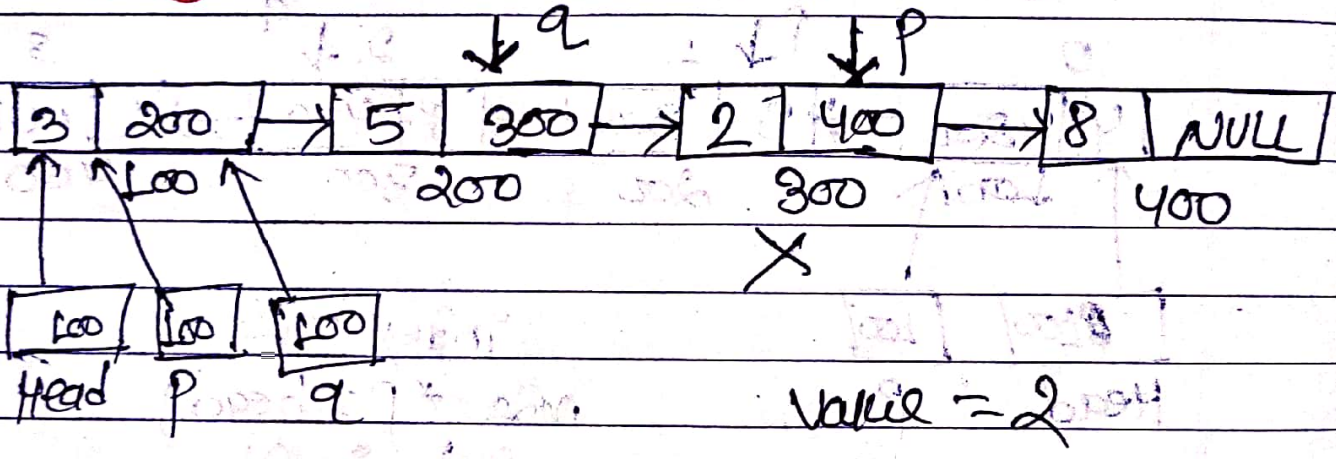
## Deleting a node in between



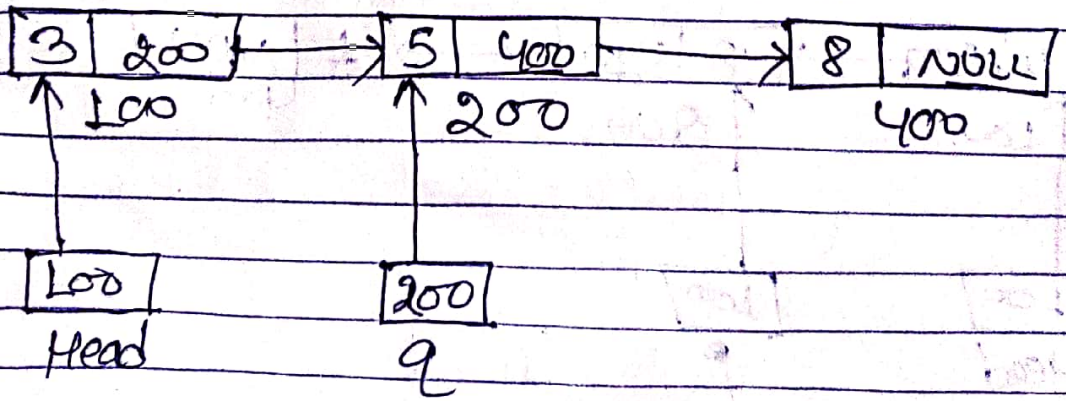

```
index = 2
node *p = head;
int i = 0;
while ( i != index-1) {
    p = p->next;
    i++;
}
node *q = p->next;
p->next = q->next;
free(q);
```



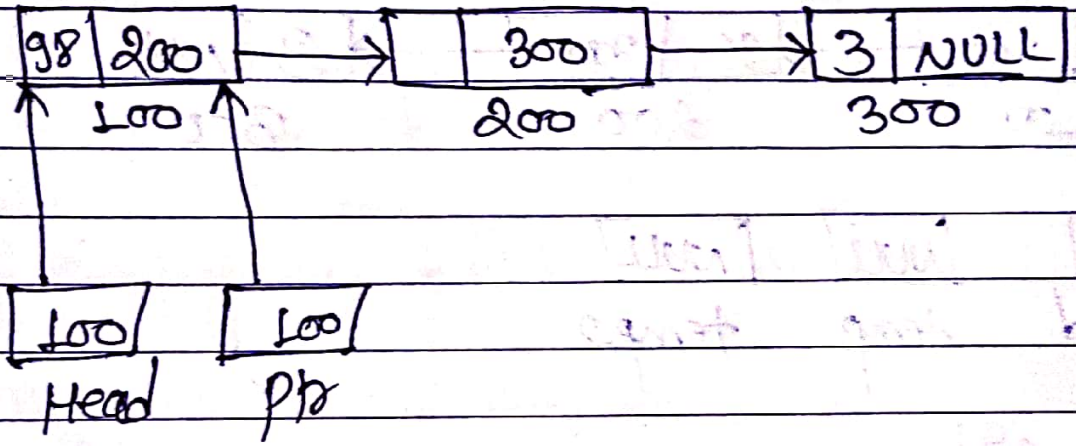
# Deleting a node with given value.




```
while(value != p->data && p->next != NULL)
{
    q = p;
    p = p->next;
}
q->next = p->next;
free(p);
```

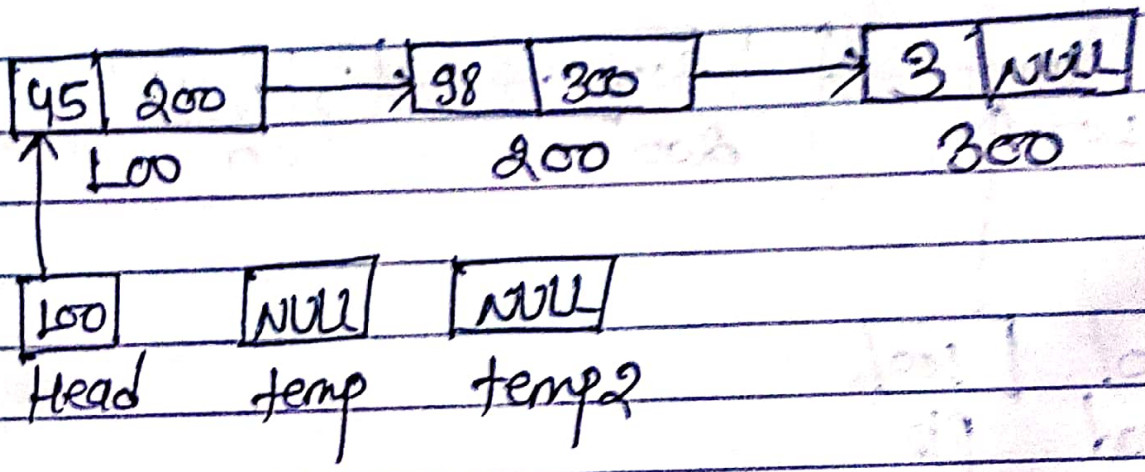


# Deleting entire linked list

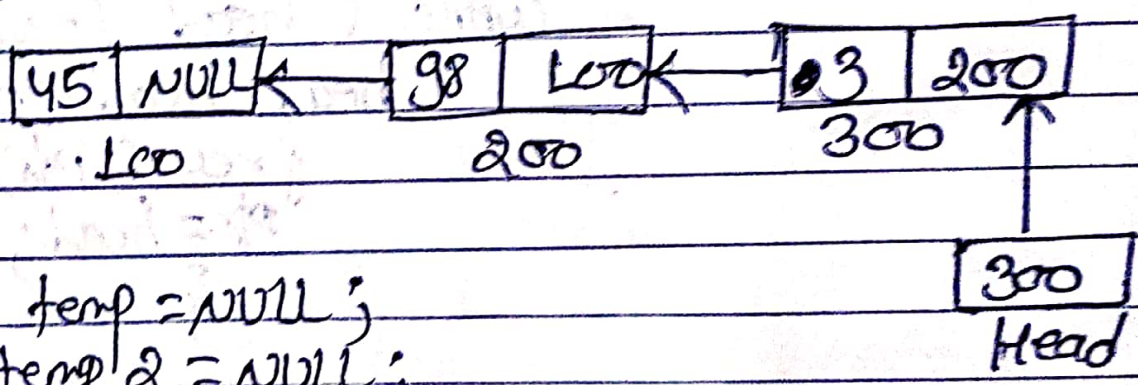



```
while ( head != NULL ){
    head = head -> next
    free (ptr);
    ptr = head;
}
```

# Reverse a linked list




New-list

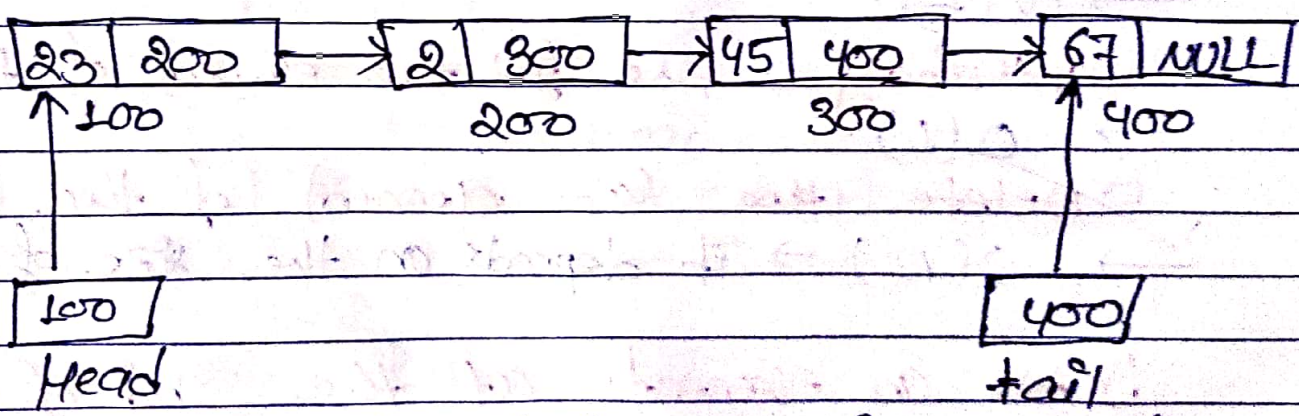



```
temp = NULL;
temp2 = NULL;
while (head != NULL) {
    temp2 = head -> next;
    head -> link = temp;
    temp = head;
    head = temp2;
}
head = temp;
```

Q. Consider an implementation of unsorted singly linked list. Suppose it has its representation with a head and a tail pointer (i.e. pointers to the first and last nodes of the linked list). Given the representation, which of the following operation can not be implemented in O(1) time?

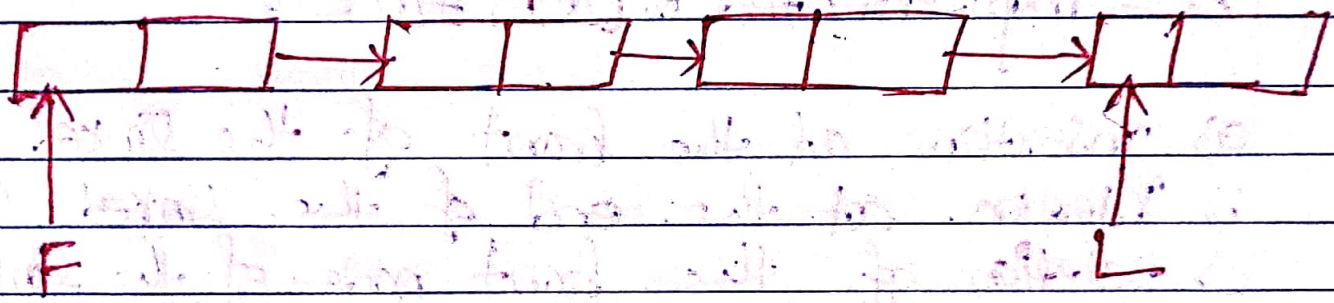
a) Insertion at the front of the linked list
b) Insertion at the end of the linked list
c) Deletion of the front node of the linked list.
d) Deletion of the last node of the linked list.

Sol:-




To delete the last node of linke list, we need the add. of the second last node. So, we have to traverse the whole list. This will take O(n) time.

Q: Consider a single linked list where F & L are pointers to the first and last element respectively of the linked list. The time for performing which of the given operations depends on the length of the linked list?




a) Delete the first element of the list
→ O(1)

b) Interchange the first two element of the list.
→ O(1)

c) Delete the last element of the list.
→ O(n) → it depends on the size of linked list

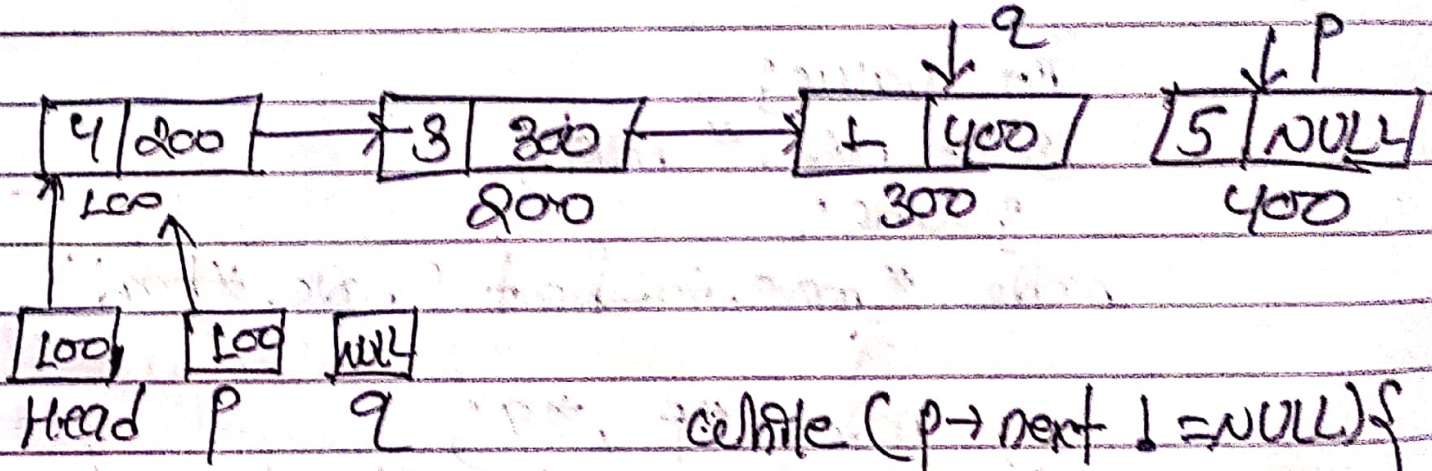
d) Add an element at the end of the list
→ O(1)

Q: The following C function takes a singly linked list as input argument. It modifies the list by moving the last element to the front of the list and returns modified list. Some part of the code is left blank.

```
typedef struct node
{
int value;
struct node * next;
} node;
node * move-to-front (node *head)
{
  node *p, *q;
  if (head == NULL) || (head -> next == NULL))
      return head
  q = NULL; p = head;
  while (p -> next != NULL)
  {
      q = p;
      p = p -> next;
  }
  ____________
  return head;
}
```

Choose the correct alternative to replace the blank line.

a) q = NULL; p->next = head; head = p;
b) q->next = NULL; head = p; p->next = head;
c) head = p; p->next = q; q->next = NULL;
d) q->next = NULL; p->next = head; head = p;




while (p->next != NULL){
q = p;
p = p->next;
}

q = 100 → 200 → 300
p = 200 → 300 → 400

q->next = NULL;
p->next = head;
head = p;

So, Correct option is 'd'. In option is sequence is not followed.

Q. The following C functions take a single-linked list of integers as a parameter and rearranges the elements of the list. The function is called with the list containing the integers 1, 2, 3, 4, 5, 6, 7 in the given order. What will be the contents of the list after function completes execution?

```
struct node {
    int value;
    struct node *next;
};
void rearrange (struct node *list) {
    struct node *p, *q;
    int temp;
    if (!list || !list->next) return;
    p = list; q = list->next;
    while (q) {
        temp = p->value; p->value = q->value;
        q->value = temp; p = q->next;
        q = p ? p->next : 0;
    }
}
```

a) 1, 2, 3, 4, 5, 6, 7
b) 2, 1, 4, 3, 6, 5, 7
c) 1, 3, 2, 5, 4, 7, 6
d) 2, 3, 4, 5, 6, 7, 1.

!list is equivalent to list == NULL

!list->next is equivalent to list->next == NULL