Linked Lists

Linked Lists

- **Linked list** is a data structure in which objects are arranged in a linear order
  - Linear order is determined by a pointer in each object

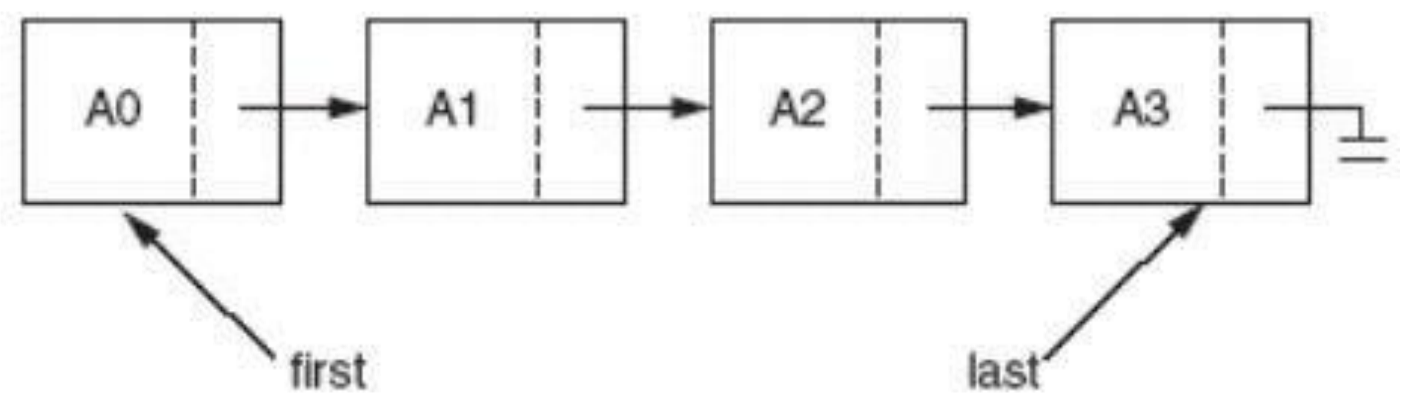
- **Different types of linked list**:
  - Singly Linked List (SLL)
  - Doubly Linked List (DLL)
  - Circular Linked List (CLL)

Linked List

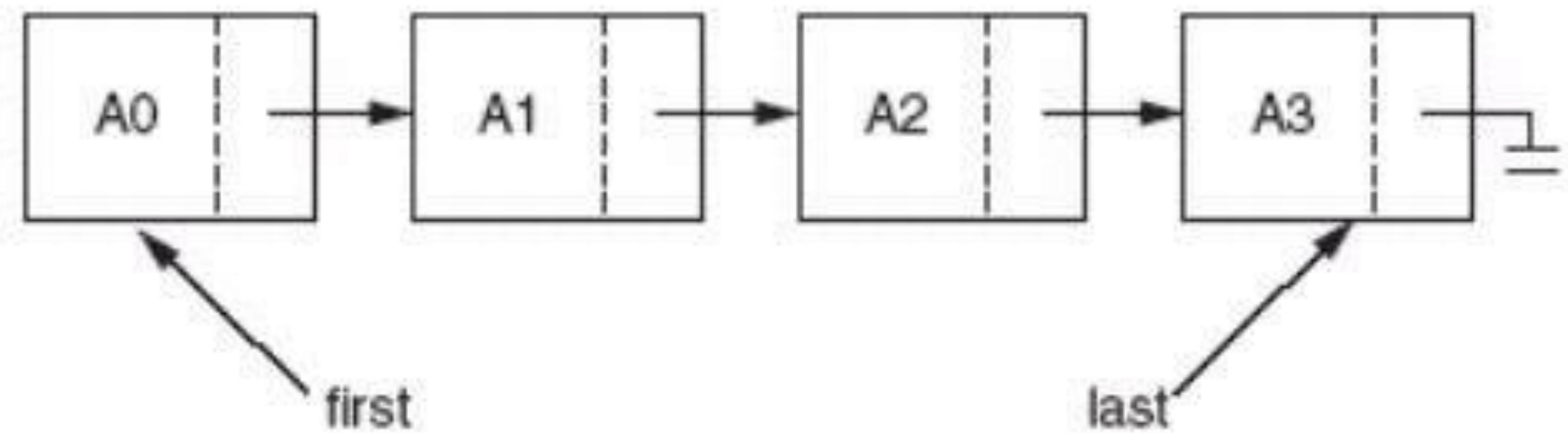
- A linked list is simply a chain of structures which contain a pointer to the next element and it is dynamic in nature.

- Items may be added to it or deleted from it.

- A list item has a pointer to the next element, or NIL if the current element is the tail (end of the list).



Example Linked List

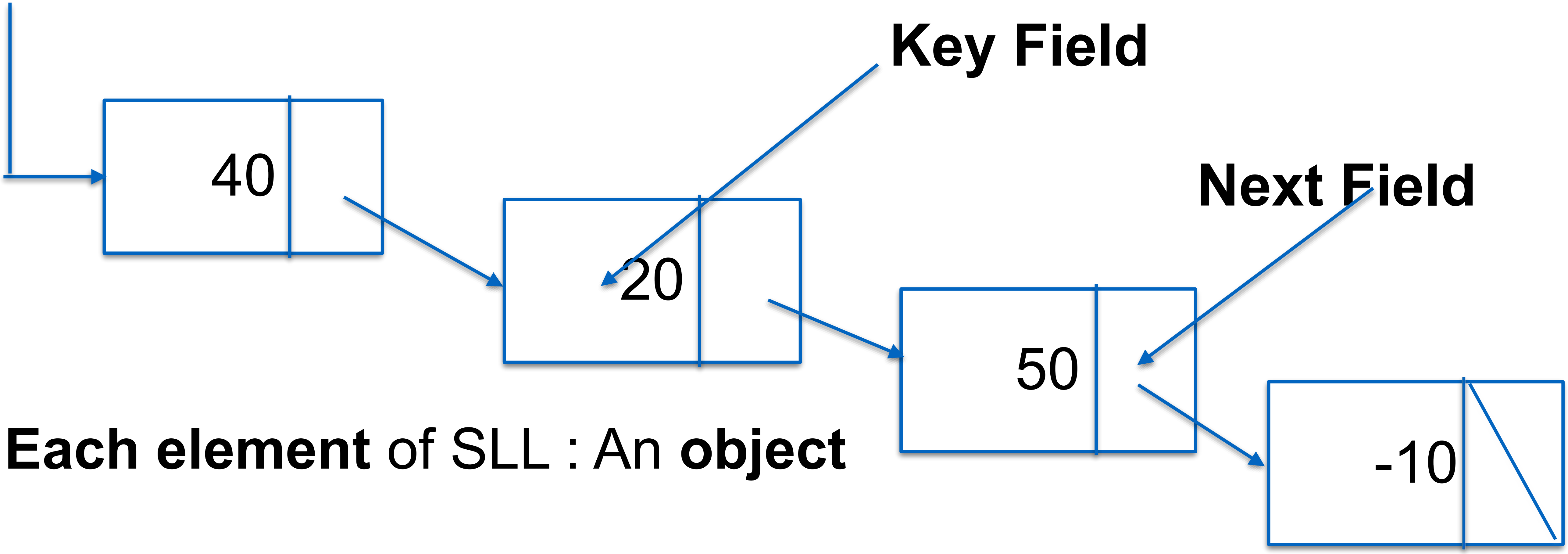


- This pointer (pointer to the next element) points to a structure of the same type as itself.

- This structure that contains elements and pointers to the next structure is called a Node.

- The first node is always used as a reference to traverse the list and is called HEAD. The last node points to NULL.

SINGLY LINKED LIST

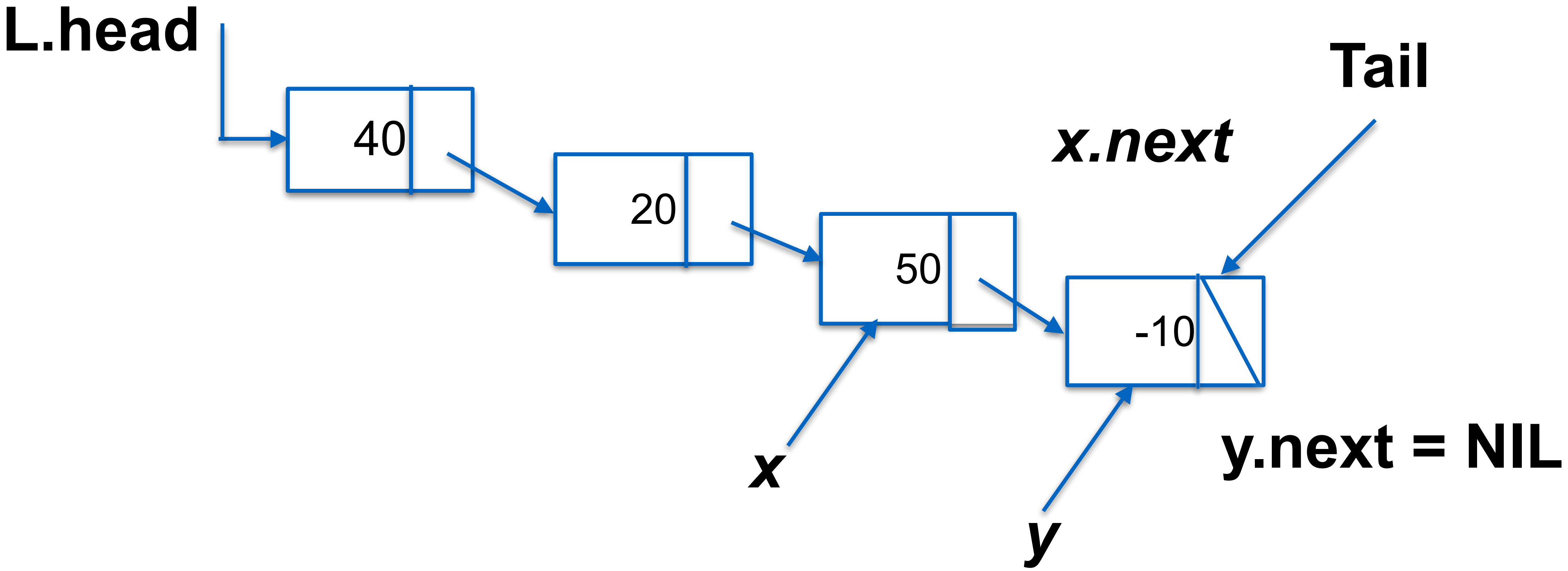


Each element of SLL : An **object**

Attributes: Key and a Next pointer

Object may also contain other **satellite data**

SINGLY LINKED LIST



- An attribute **L.head** points to the **first element** of the list.
  If **L.head = NIL, the list is empty**
- Given an element *x* in the list, ***x.next*** points to its **successor** in the linked list
- If **x.next = NIL**, the element x has **no successor** and is therefore the last element, or **tail**, of the list.

Declaring a node in Linked list

Declaring a node in a Linked list :

```
struct node
{
    long int key;
    struct node *next;
};
```

The above definition is used to create every node in the list.

The **key** field stores the element and the **next** is a pointer to store the address of the next node.

In place of a data type, **struct node** is written before next.

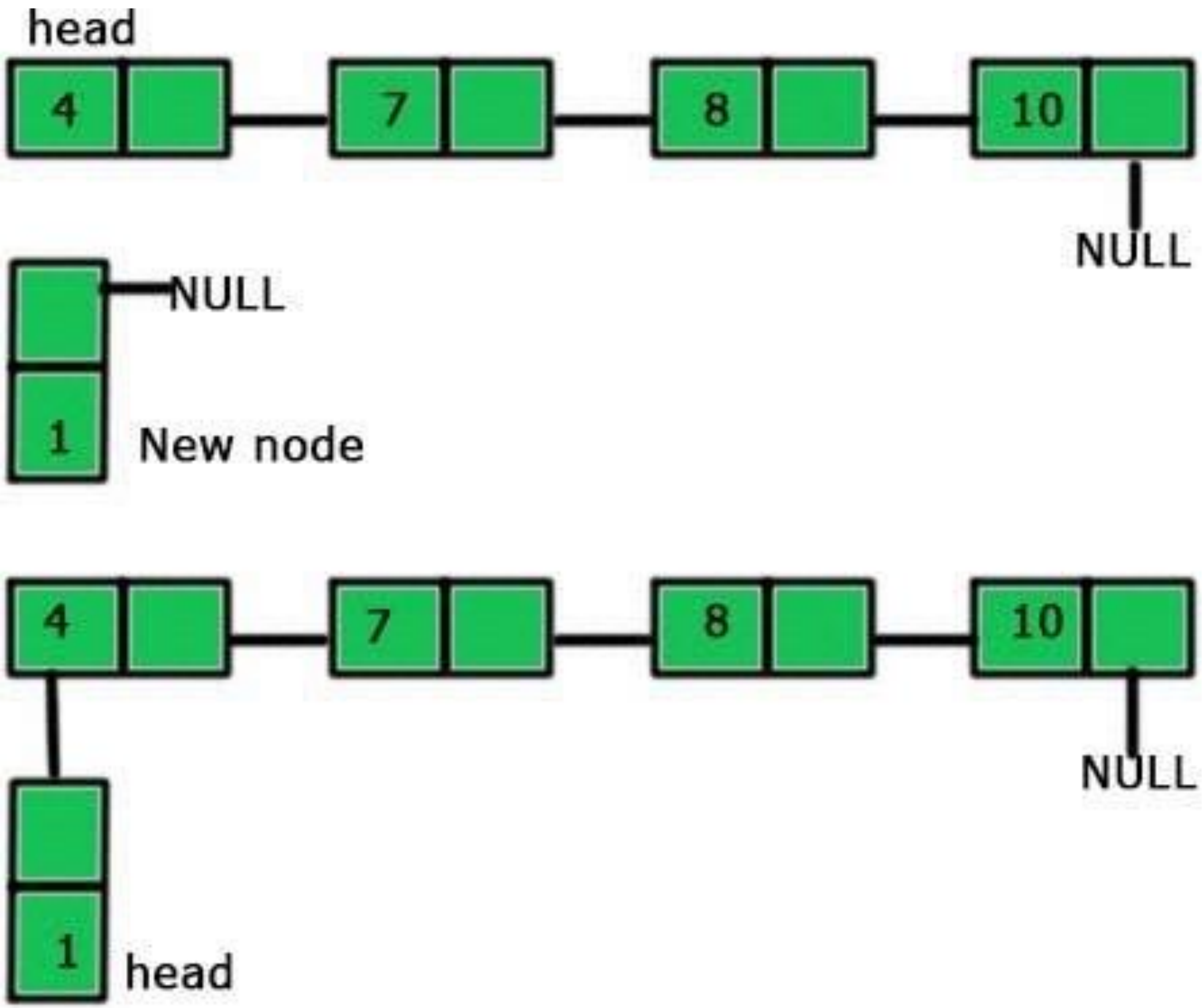
That's because its a **self-referencing pointer.** It means a pointer that points to whatever it is a part of.

Here, **next** is a part of a node in the linked list and it will point to the next

Creating a Node

```
struct node
{
    long int key;
    struct node *next;
};
typedef struct node *node; //Define node as pointer of data type struct node
struct LL //  LL stores a pointer to the head of the LL
{
    node head; // head is a pointer to the struct node
};
typedef struct LL *LL; //Define LL as pointer of data type struct LL
node CREATE_NODE(long int k)
{
    node temp;
    temp = (node)malloc(sizeof(struct node)); // allocate memory using malloc()
    if(temp == NULL)
        exit(0);
    temp->key = k;
    temp->next = NULL; // make next point to NULL
    return temp; //return the new node
}
```

Insertion at thebeginning of the linked list

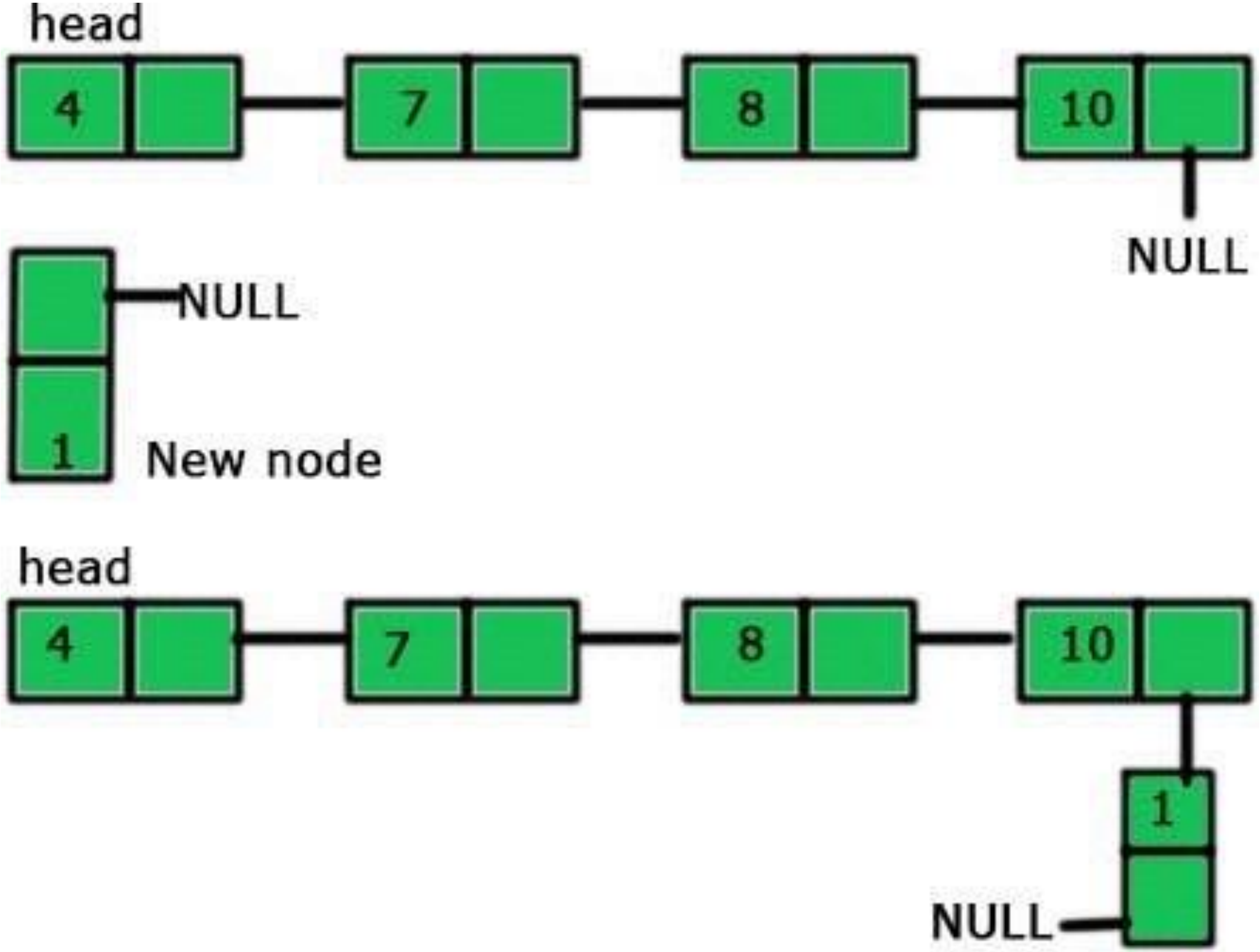


Insertion at the beginning of the linked list

Assume that the node x is created using CREATE_NODE function, **pointer to the node x is passed as an argument to the function**

```
void LIST_INSERT_FRONT(LL L, node x)
{
   x->next = L->head;
   L->head = x;
}
```

Insertion at the end of the linked list



Inserting a node to the end/tail of the linked list

Assume that the node x is created using CREATE_NODE function

```
void LIST_INSERT_TAIL(LL L,node x)
{
    node selected=L->head;
    if(selected!=NULL)
    {
        while(selected->next!=NULL)
            selected=selected->next;
        selected->next=x;
    }
    else
        L->head=x;
}
```

Inserting a node to the end/tail/rear of the linked list

- Here the new node will always be added after the last node. This is known as inserting a node at the rear end.

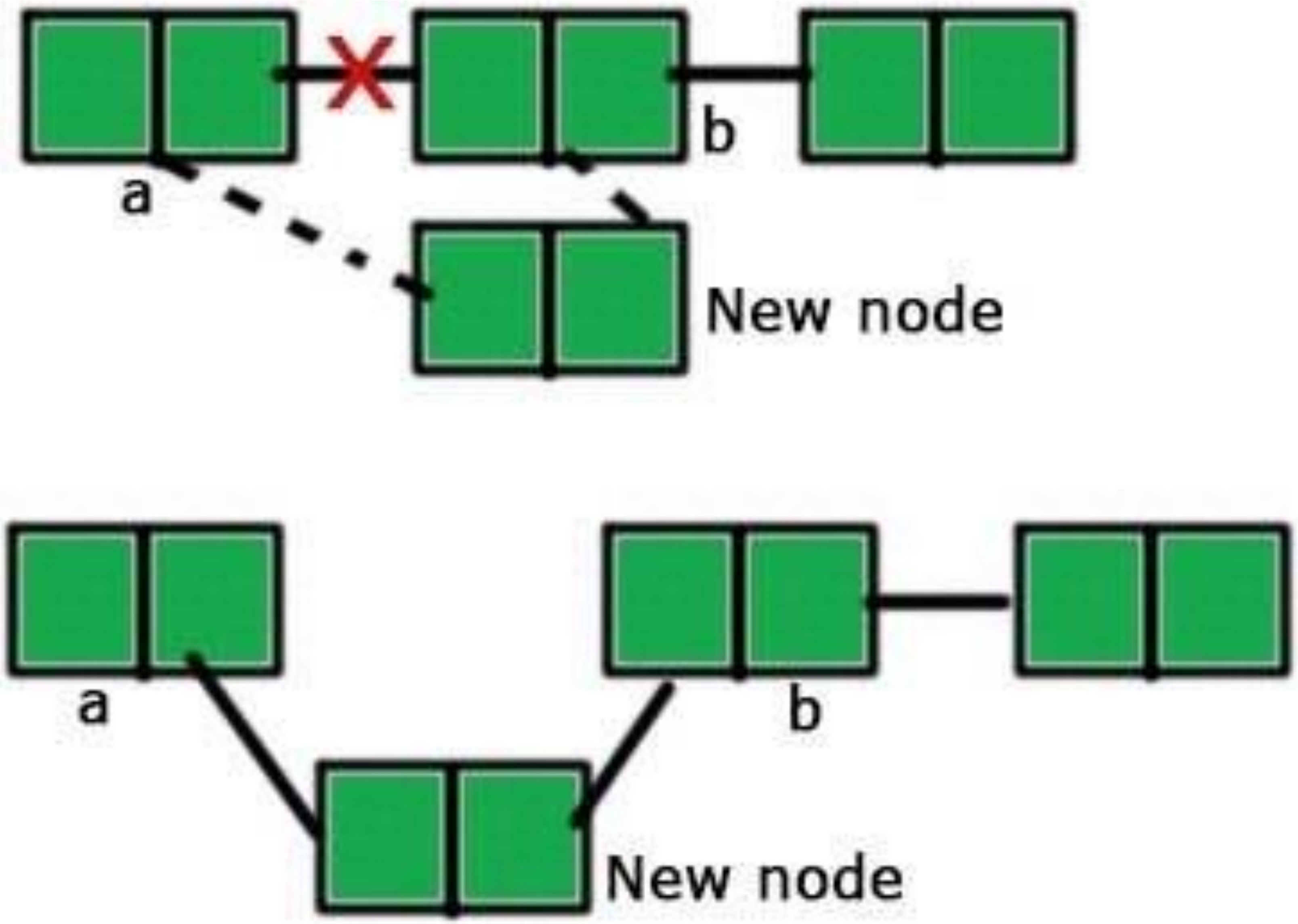
- A simple linked list can be traversed in only one direction from head to the last node.

- -> is used to access next sub element of node p. NULL denotes no node exists after the current node, i.e. its the end of the list.

Insertion at the end of the linked list

What is the running time to insert the element in the tail of the list?

Insertion in-between the linked list



LIST-INSERT at a specific position EXERCISE

Write the Pseudocode/Algorithm to insert the element x at a particular position?

What is the running time to insert the element x at a particular position the list?

Search the LL for a key k

```
node LIST_SEARCH(LL L,long int k)
{
    node selected=L->head;
    while(selected!=NULL && selected->key!=k)
        selected=selected->next;
    return selected;
}
```

Printing the linked list

```
void print(LL L)
{
  node selected=L->head;
  while(selected!=NULL)
  {
    printf("%ld-->",selected->key);
    selected=selected->next;
  }
  printf("\n");
}
```

Deletion of a node from a Linked List

- Delete a node from the front of the linked list

- Delete a node from the tail/rear of the linked list

- Delete a node from any other position than the front/ rear of the linked list

- Deleting the whole linked list

Deletion of a node: front of the Linked List

```
void LIST_DELETE_FIRST(LL L)
{
    node selected=L->head;
    if(selected==NULL)
        printf("-1\n");
    else
    {
        printf("%ld\n",selected->key);
        L->head=L->head->next;
        free(selected);
    }
}
```

Delete a node from the rear of the linked list

```
void LIST_DELETE_LAST(LL L)
{
   node selected=L->head;
   node previous=NULL;
   if(selected==NULL)
      printf("-1\n");
   else
   {
      while(selected->next!=NULL)
      {
         previous=selected;
         selected=selected->next;
      }
      printf("%ld\n",selected->key);
      if(previous!=NULL)
         previous->next=NULL;
      else
         L->head=NULL;
   }
   free(selected);
}
```

Exercises

Delete a node from any other position than the front/ rear of the linked list

Delete the whole linked list

Count the number of nodes in a linked list

Reversing the linked list

Linked list operations - Exercises

Insertion of a node (to maintain an ordered linked list)

- **Input:** Linked List and an element to be inserted
- **Output:** Ordered linked list

Deletion of a given node in the linked list

- **Input:** Linked list and an element x to be deleted
- **Output:** Ordered linked list without x