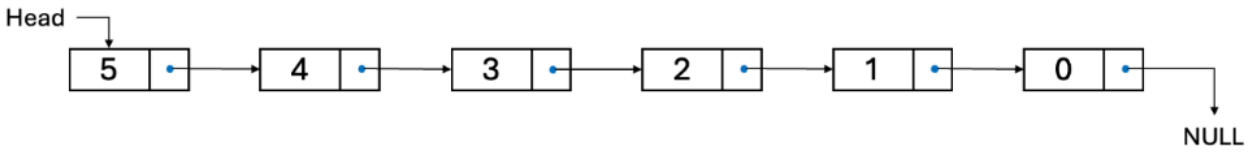
Linked List

一、核心概念

一種**線性資料結構**，由一系列的**節點 (Node)** 組成。



1. 節點結構

- **資料欄位 (Data Field)：** 儲存實際的數值（如整數、字串等）。

- **指標欄位 (Pointer Field)：** 儲存下一個節點的記憶體位址，通常稱為 next。

2. 記憶體特性

與陣列不同，鏈結串列的節點儲存在**非連續的記憶體空間**中。它們透過「指標」像火車車廂一樣串連起來。

- **優點**：大小可動態增減，不需要預先宣告固定長度。

- **缺點**：不能像陣列那樣透過索引 (Index) 直接存取，必須從頭節點 (Head) 開始逐一走訪。

二、 Linked List vs. Array

1. 靜態 vs. 動態比較表

特性	Array	Linked List
記憶體分配	連續空間 (Contiguous)	非連續空間 (Non-contiguous)
大小限制	固定大小 (Static/Fixed)	動態調整 (Dynamic)
存取元素	隨機存取 O(1)	循序存取 O(n)
插入/刪除	較慢 O(n)（需移動元素）	較快 O(1)（若已知位置）
額外開銷	極低	較高（需儲存指標空間）

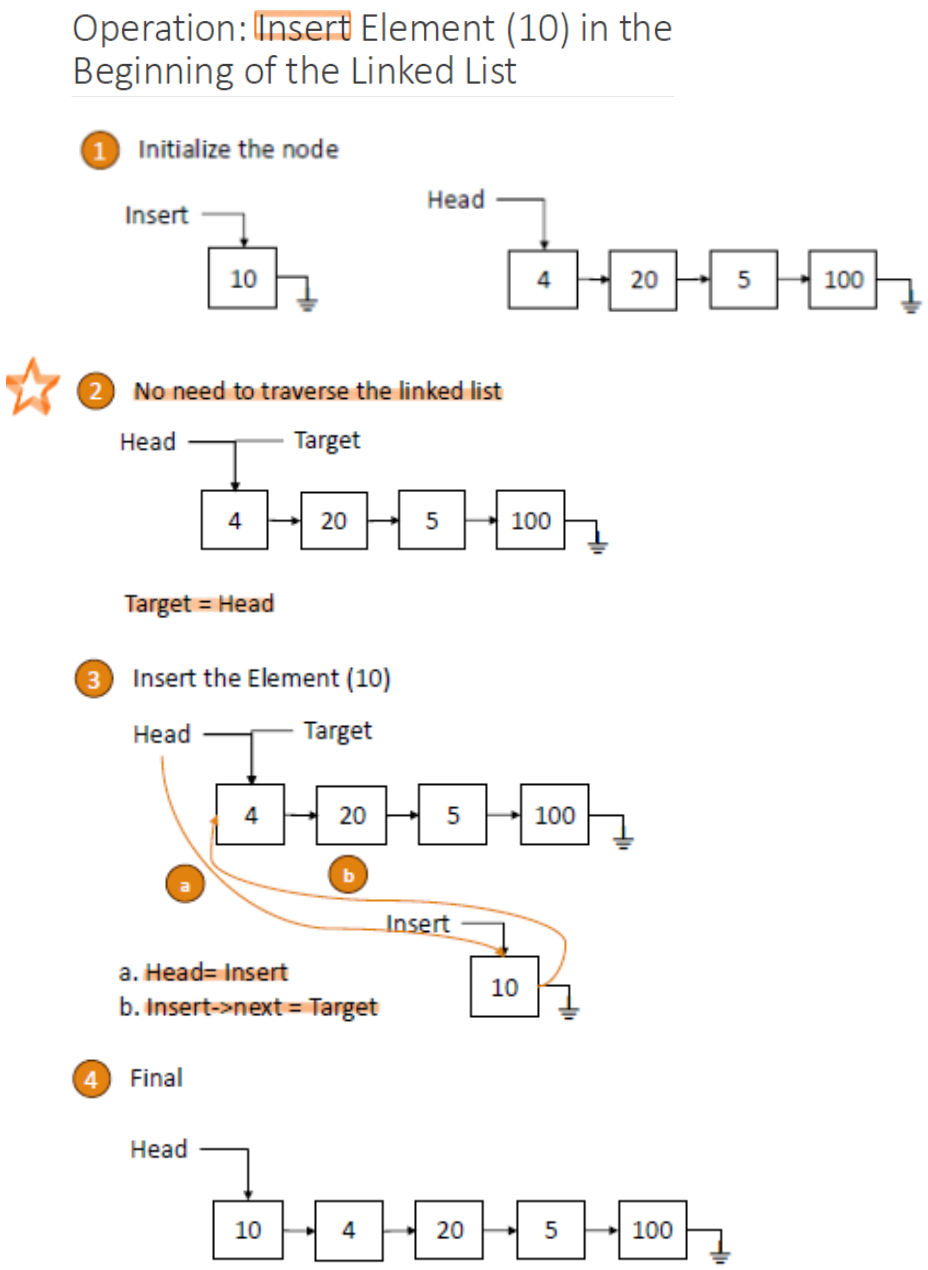
三、 基本操作邏輯 (C++)

1. 節點定義

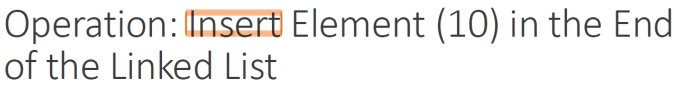
```
class Node {
public:
    int data;
    Node* next;
    Node(int val) : data(val), next(nullptr) {}
};
```

2. 插入 (Insertion)

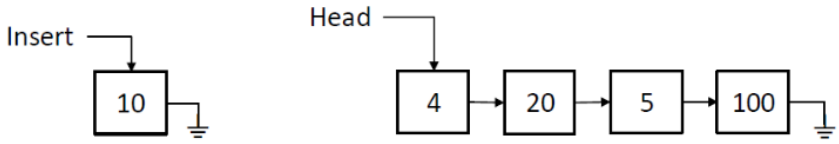
- **頭部插入：** 將新節點的 next 指向目前的 head，再將 head 更新為新節點 。



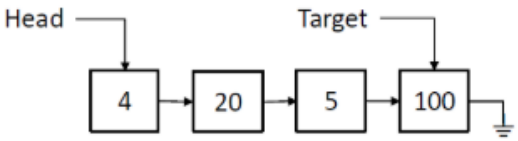
- **尾部插入：** 需先走訪到最後一個節點（其 next 為 NULL），再將其 next 指向新節點 。



1 Initialize the node

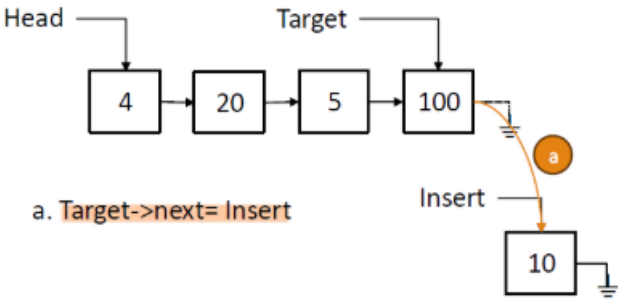


2 Traverse the linked list to find the end



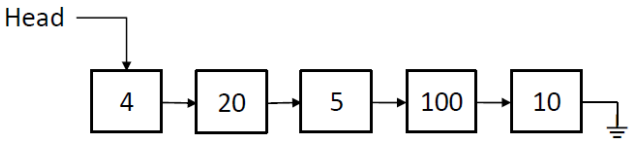

```
Target = Head
While (Target->next != NULL){
      Target = Target->next;
}
```

3 Insert the Element (10)



a. Target->next= Insert

4 Final



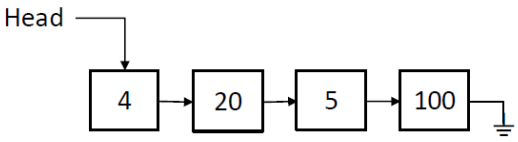
3. 刪除 (Deletion)

關鍵在於找到待刪除節點的前一個節點 (Prev)。

- **操作：Prev → next = Target → next; 接著釋放 (free/delete) Target 的記憶體以避免記憶體洩漏 (Memory Leak) 。**
- **頭部刪除:**

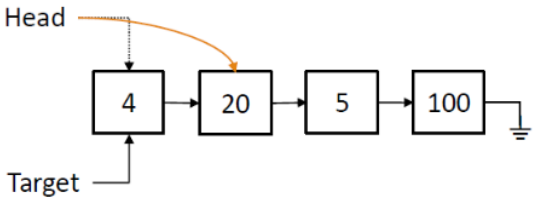
Operation: Delete the Target (First)

1 No need to traverse the linked list



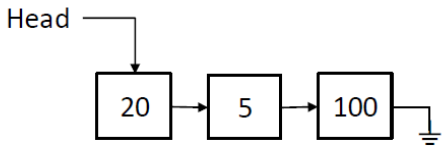
Target = Head
Prev = Head

2 Delete the first element (4)



Head = Target->next
free(target) 釋放.

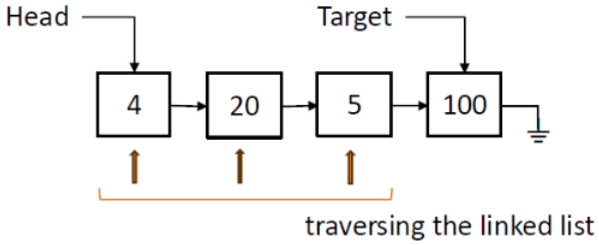
3 Final



- **尾部刪除**

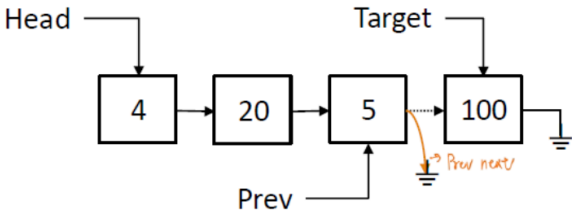
Operation: Delete the Target (Last)

1 Traverse the linked list to fine the end



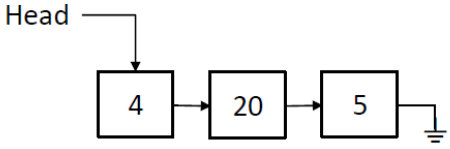

```
Target = Head
Prev = Head

While (Target->next !== NULL){ Target next 為 Null 時 break，執行下一步，
    Prev = Target
    Target = Target->next
}
```

2 Delete the first element (4)




```
Prev->next = NULL
free(target) 釋放.
```

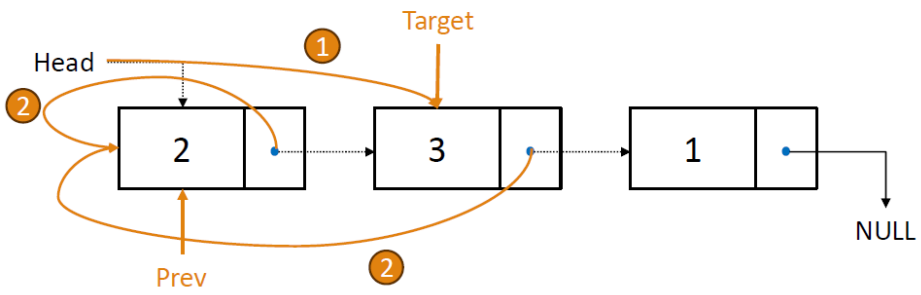
3 Final



4.Move To

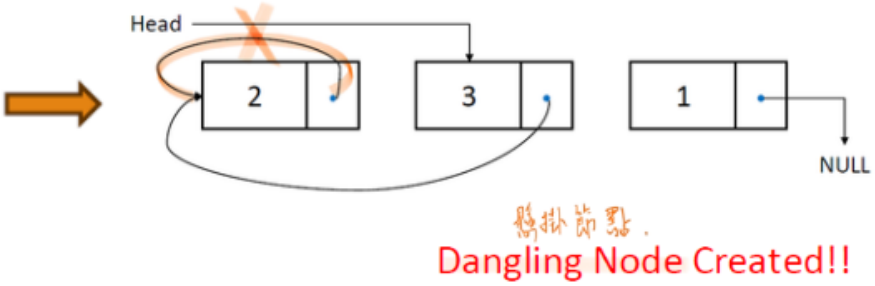
在調整指標順序時，若順序錯誤可能會產生 **懸掛節點 (Dangling Node)**，導致後續節點無法存取 。

- **Wrong Case**

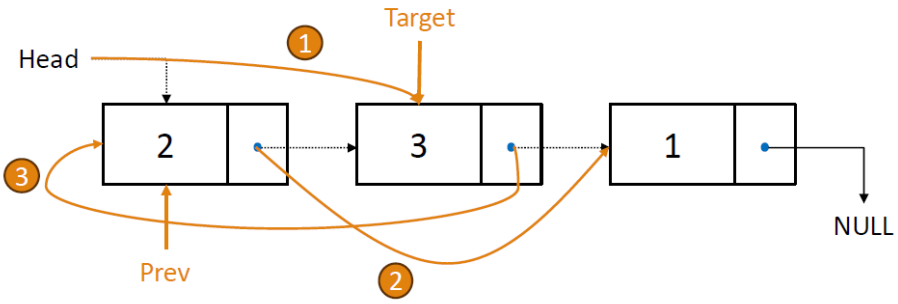


Prev: Prev
Prev->next
Target: Target
Target->next

Steps
1. Head = Prev → Head = Target
2. Target->next = Prev
3. Prev->next = Target->next (Prev)

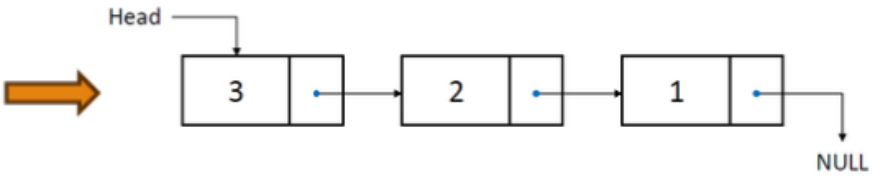


- **Correct Case**



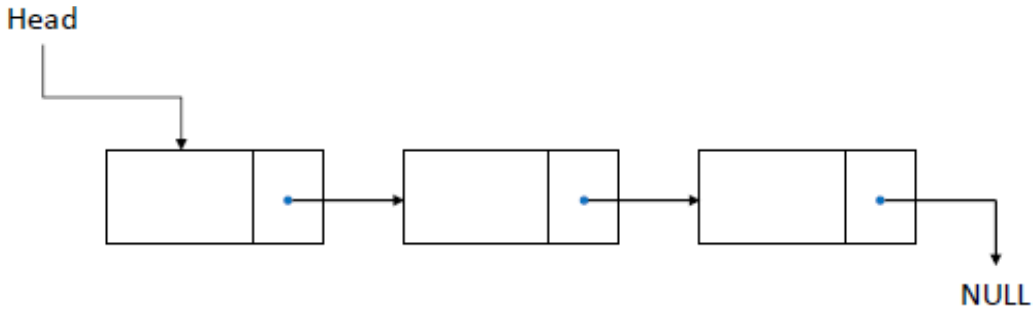
Prev: Prev
Prev->next
Target: Target
Target->next

Steps
1. Head = Prev → Head = Target
2. Prev->next = Target->next
3. Target->next = Prev



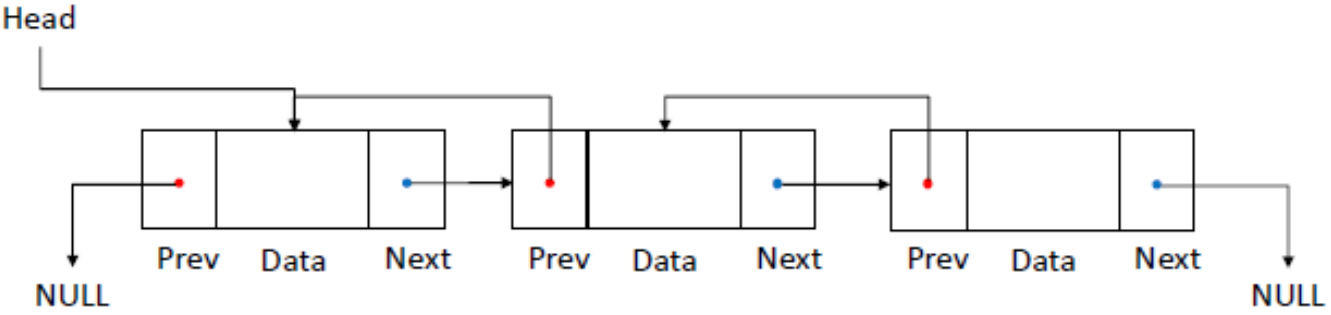
四、進階變體

1.單向鏈結串列 (Singly Linked List)： 基礎形式，只能向後走訪 。



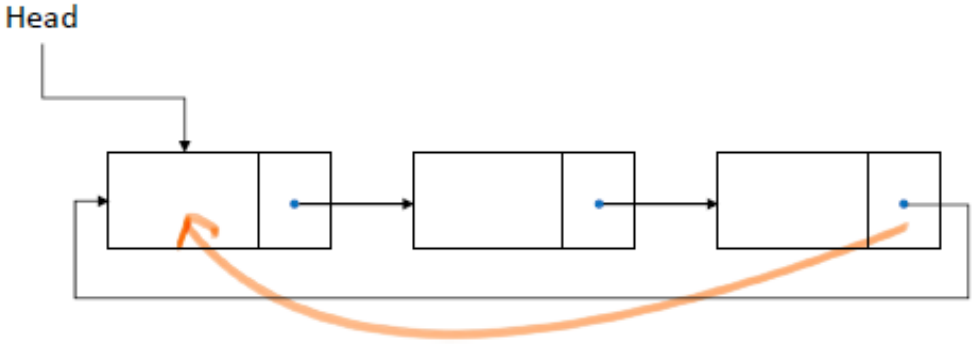
2.雙向鏈結串列 (Doubly Linked List)： 每個節點多了一個 prev 指標指向前一個節點 。

- 優點： 可以雙向導覽，在已知節點的情況下，刪除操作更有效率 。



3.環狀鏈結串列 (Circular Linked List)： 最後一個節點的 next 指回 head 。

- 應用： 多人遊戲輪流機制、作業系統排程 。



五、選擇排序法 (Selection Sort)

在鏈結串列上執行選擇排序有兩種方式：

1. 交換數值 (Swap Value)

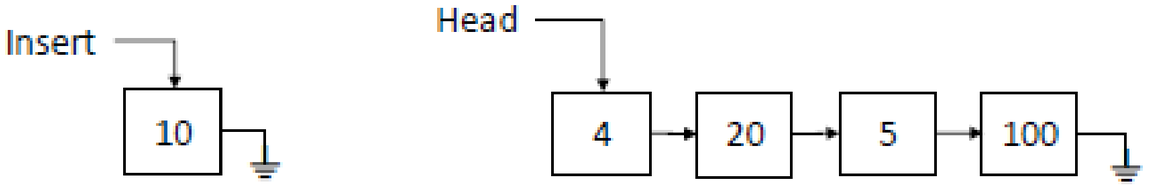
- 邏輯： 找到未排序部分的最小值，將其與當前節點的 data 交換 。
- 優點： 實作簡單，指標不動 。

- 缺點： 可能是不穩定排序 (Unstable) 。

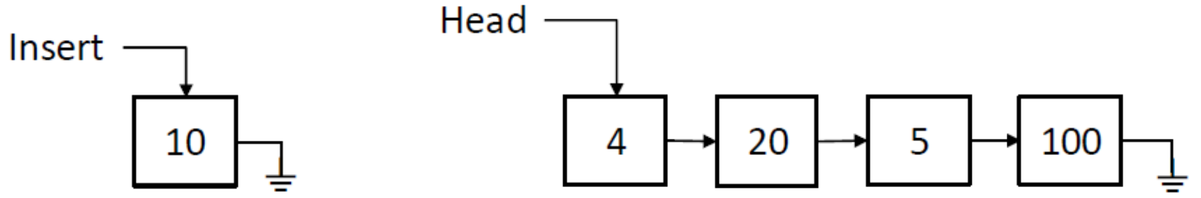
2. 交換指標 (Swap Pointer/Node)

- 邏輯： 重新調整節點間的鏈結關係，將節點「移動」到正確位置 。
- 優點： 真正發揮鏈結串列「移動節點不需搬移資料」的優勢 。
- 常用技巧： 使用 虛擬頭節點 (Dummy Node) 來簡化頭部變動的邊界處理 。

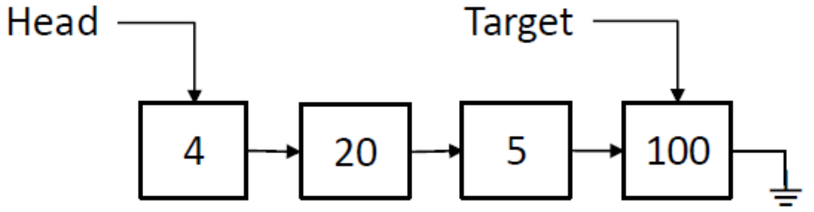
1 Initialize the node



1 Initialize the node

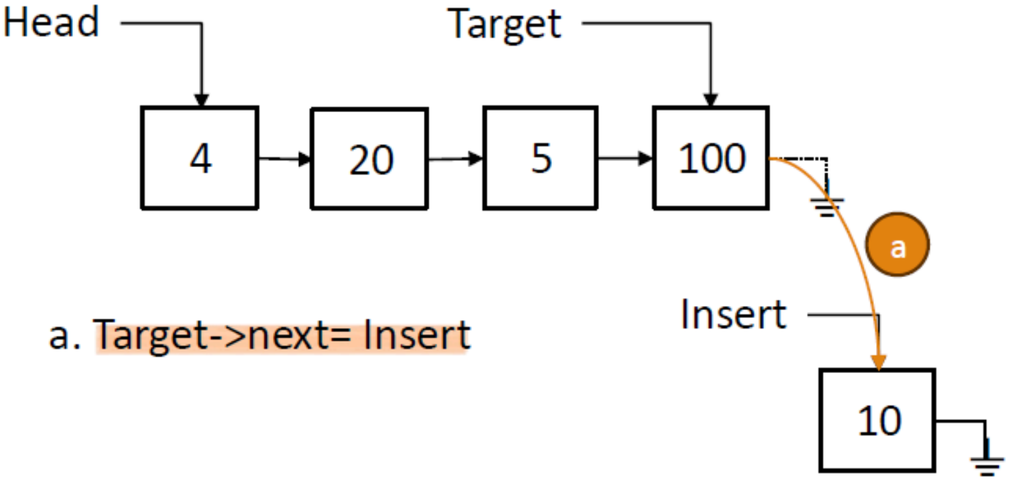


2 Traverse the linked list to find the end



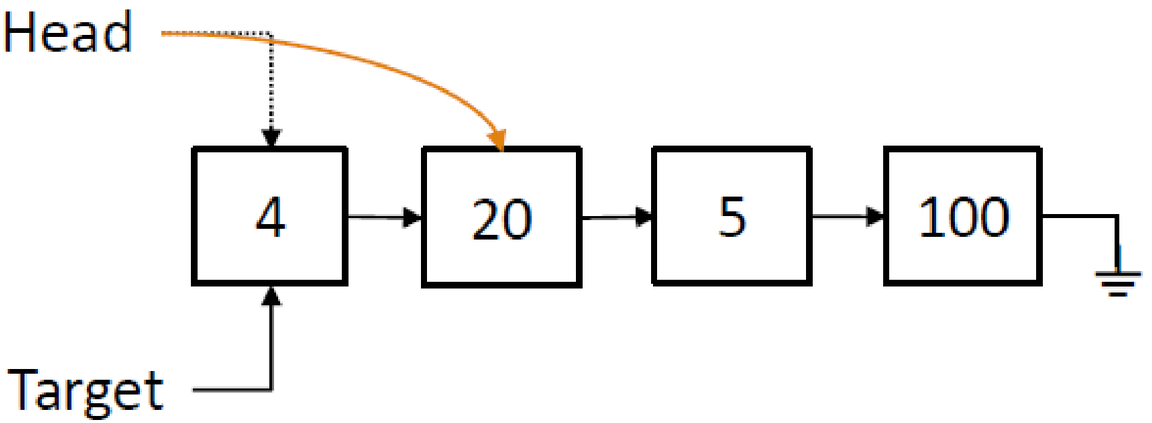

```
Target = Head → 從頭開始。
While (Target->next != NULL){ → Target next 為 Null 時 break，執行下一步。
      Target = Target->next;
}
```

3 Insert the Element (10)



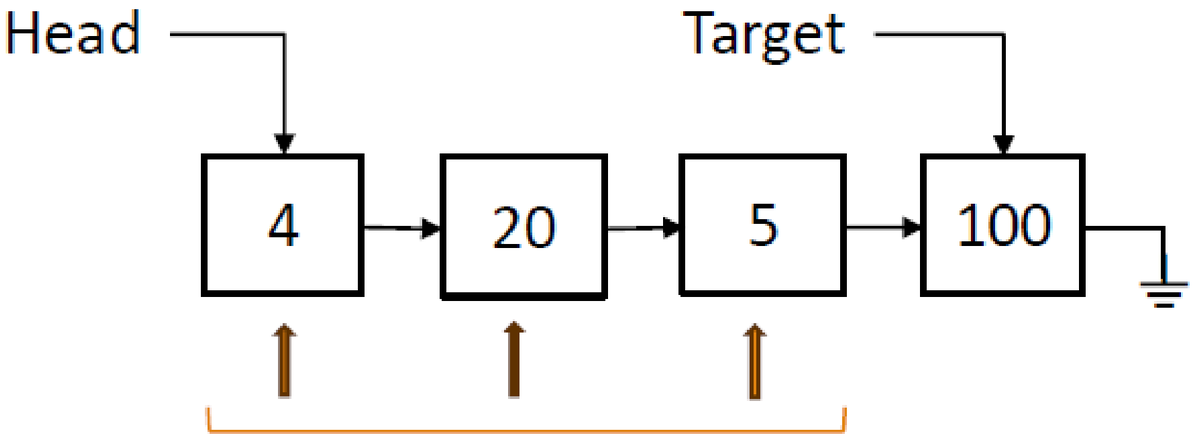
a. Target->next= Insert

2 Delete the first element (4)



Head = Target->next
free(target) 釋放.

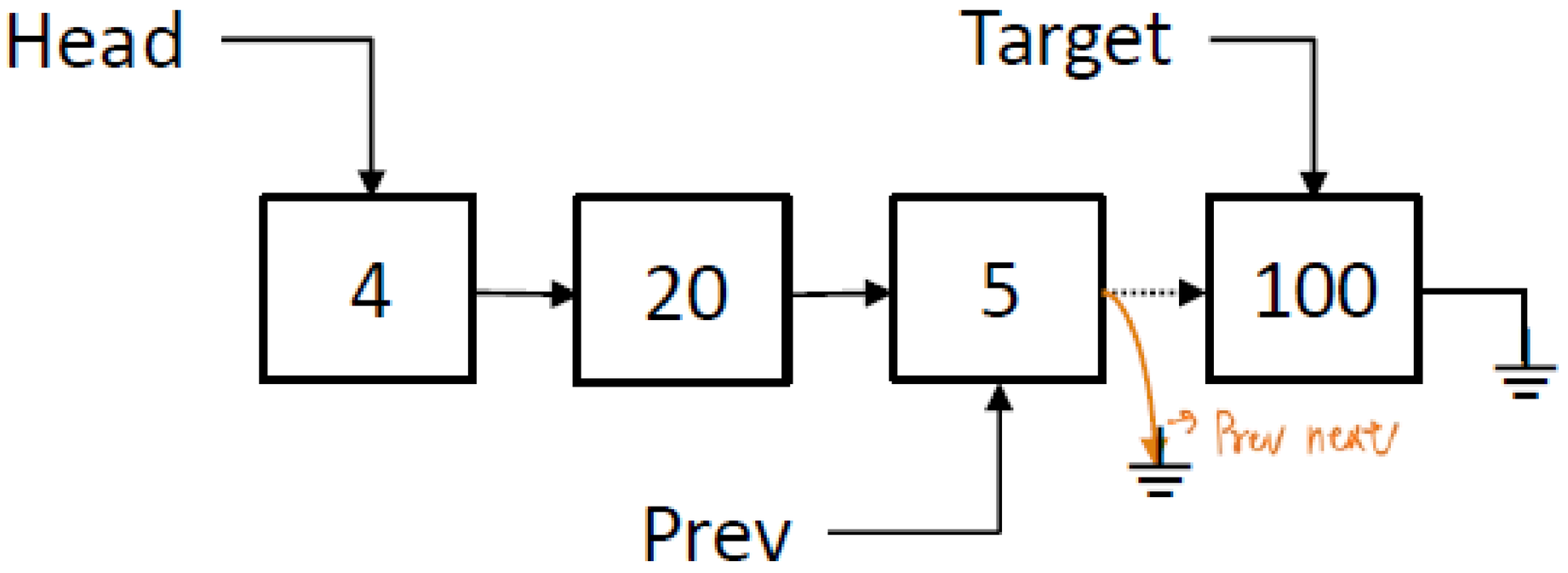
1 Traverse the linked list to fine the end




```
Target = Head
Prev = Head

While (Target->next !== NULL){
        Prev = Target
        Target = Target->next
}
```

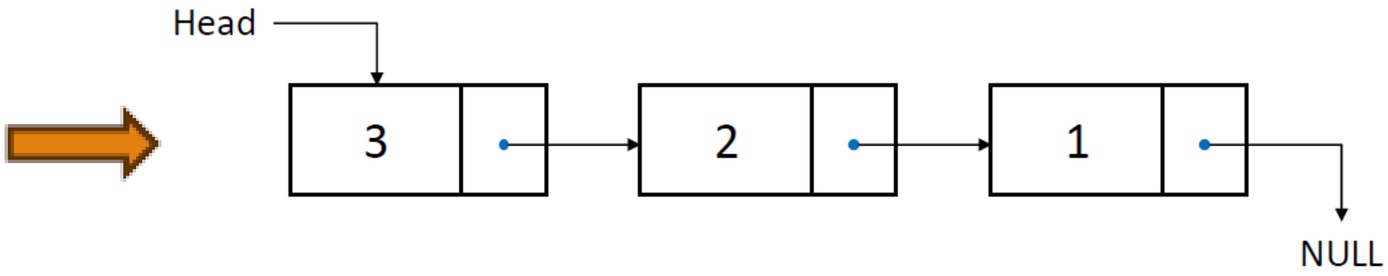
→ Target next 为 Null 时 break，执行下一步。

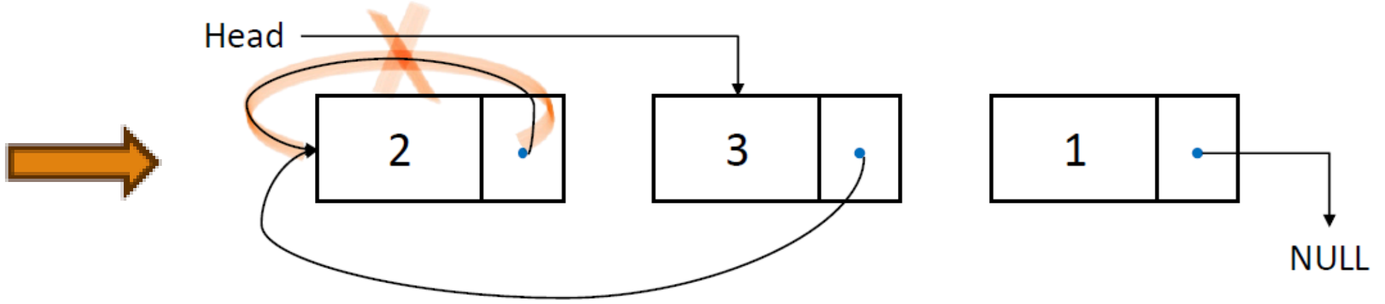
2 Delete the first element (4)



Prev->next = NULL

free(target) 釋放.





懸掛節點 .

Dangling Node Created!!