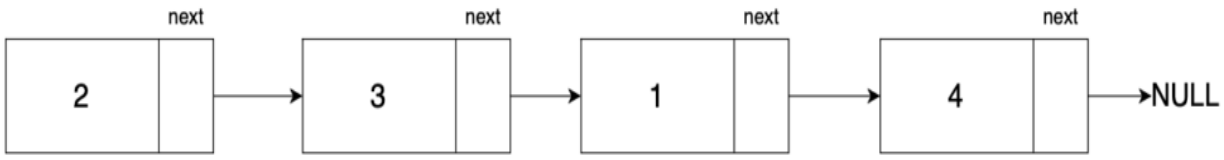
Introduction to Doubly Linked List

In the realm of data structures, it's essential to grasp the intricacies of doubly linked lists. These data structures are characterized by their ability to efficiently **navigate in both forward and backward directions**. Before diving into the depths of doubly linked lists, it's beneficial to recap our understanding of linked lists, and their precursor.

Recap on Linked List



A singly linked list

Before exploring doubly linked lists, let's refresh our knowledge of linked lists. Linked lists are linear data structures consisting of nodes, **each containing data and a reference (or pointer) to the next node.** This setup allows for dynamic memory allocation and efficient insertions and deletions.

A significant characteristic of singly linked lists is their **unidirectional nature**, allowing **traversal in only one direction**: **forward**. Moving backward, such as going from node 1 to node 3, is not possible because each node in a singly linked list holds two pieces of information - **the data** (an integer value in this case) and **a pointer** that indicates the address **of the next node**. This structure enables efficient forward navigation, but the absence of a backward pointer restricts reverse traversal.

Doubly Linked Lists, as the name suggests, take the concept of **2-way traversal** by **introducing two pointers in each node**. This **enables seamless traversal in both directions**, making them a valuable tool for various advanced data structure applications.

Code:

```
class Node {
public:
    int data;       // Data stored in the node
    Node* next;     // Pointer to the next node in the list

    // Constructor for a Node with both data and a reference to the next node
    Node(int data1, Node* next1) {
        data = data1;
        next = next1;
    }

    // Constructor for a Node with data and no reference to the next node (end
of the list)
    Node(int data1) {
        data = data1;
        next = nullptr;
    }
};
```

The code for Singly Linked List creates a class Node which has two **member variables:**

- **int data:** This holds the data value that the node stores, which can be of any data type (in this case, an integer).
- **Node* next:** This is a pointer to the next node in the linked list. It allows the nodes to be linked together, forming a sequence.

The class has two **constructors**:

- The first constructor takes **both the data and a pointer to the next node** as parameters. It initializes the data and the next node accordingly.
- The second constructor takes only the **data as a parameter** and **sets it next to nullptr.** It could be that this node is the end of the list since there is no reference to the next node. We can append the next of this node later **adding more nodes to the linked list.**

Here is the code for the doubly linked list

Code:

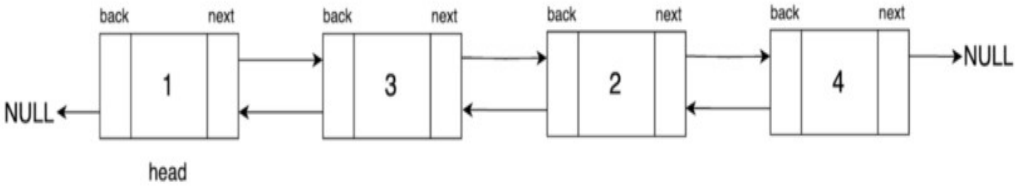
```
class Node {
public:
    int data;       // Data stored in the node
    Node* next;     // Pointer to the next node in the list (forward direction)
    Node* back;     // Pointer to the previous node in the list (backward
direction)

    // Constructor for a Node with both data, a reference to the next node, and
a reference to the previous node
    Node(int data1, Node* next1, Node* back1) {
        data = data1;
        next = next1; // Similar to a singly linked list, but now with a 'back'
pointer.
        back = back1; // Addition of 'back' pointer for the doubly linked list.
    }

    // Constructor for a Node with data, and no references to the next and
previous nodes (end of the list)
    Node(int data1) {
        data = data1;
        next = nullptr; // Similar to a singly linked list.
        back = nullptr; // Addition of 'back' pointer initialization.
    }
};
```

- **Node* back; :** The introduction of the back pointer is the key change from a singly linked list node. This pointer allows **traversal in the backward direction**, making it suitable for doubly linked lists.
- **Constructors:** Both constructors have been updated to initialize the new back pointer. In the first constructor, Node(int data1, Node* next1, Node* back1), back is initialized with the provided value. In the second constructor, Node(int data1), **the back is initialized to nullptr, just like the next.**

These changes differentiate the Node class for a doubly linked list, allowing it to maintain **bidirectional links between nodes**, as opposed to the unidirectional links in a singly linked list node.



A doubly linked list