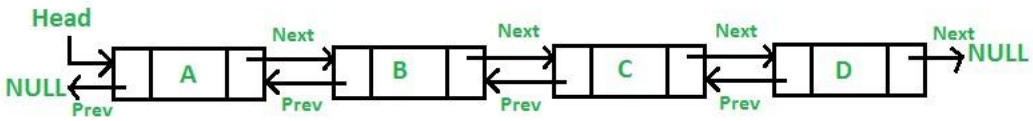
Doubly Linked List

Similar to *Singly Linked Lists*, ***Doubly Linked Lists*** are also a sequential data structure with the only difference that the doubly linked lists contain two pointers instead of one to store the address of both next node and previous node respectively.



As you can see in the above image:

- Each node contains two pointers, one pointing to the next node and the other pointing to the previous node.
- The prev of Head node is NULL, as there is no previous node of Head.
- The next of last node is NULL, as there is no node after the last node.

Below is the sample Doubly Linked List node in C++ and Java:

C++ Java

```cpp

/* Node of a doubly linked list */
struct Node {
    int data;
    struct Node* next; // Pointer to next node in DLL
    struct Node* prev; // Pointer to previous node in DLL
};

```

Advantages of doubly linked lists over singly linked list:

1. A DLL can be traversed in both forward and backward direction.
2. The delete operation in DLL is more efficient if the pointer to the node to be deleted is given.
3. We can quickly insert a new node before a given node.
4. In a singly linked list, to delete a node, a pointer to the previous node is needed. To get this previous node, sometimes the list is traversed. In DLL, we can get the previous node using the previous pointer.

Disadvantages of doubly linked lists over singly linked list:

1. Every node of DLL Require extra space for a previous pointer.
2. All operations require an extra pointer previous to be maintained. For example, an insertion, we need to modify previous pointers together with next pointers.

Creating and Traversing a Doubly Linked List

Creating and Traversing a doubly linked list is almost similar to that of the singly linked lists. The only difference is that in doubly linked lists we need to maintain an extra previous pointer for every node while creating the list.

Below is the complete program to create and traverse a Doubly Linked List in both C++ and Java:

C++ Java

```cpp
// A complete working C++ program to demonstrate Doubly Linked Lists
#include <bits/stdc++.h>
using namespace std;
// A linked list node
struct Node {
    int data;
    struct Node* next;
    struct Node* prev;
};
// This function prints contents of Doubly linked
// list starting from the given node
void printList(struct Node* node)
{
    struct Node* last;
    // Traverse the linked list in forward direction
    // using the next node's pointer present at each node
    cout<<"\nTraversal in forward direction \n";
    while (node != NULL) {
        cout<<node->data<<" ";
        last = node;
        node = node->next;
    }
// Traverse the linked list in reverse direction starting from the last node
//using the previous node's pointer present at each node
    printf("\nTraversal in reverse direction \n");
    while (last != NULL) {
        cout<<last->data<<" ";
        last = last->prev;
    }
}

/* Given a reference (pointer to pointer) to the head
   of a DLL and an int, this function inserts
   a new node at the end */
void append(struct Node** head_ref, int new_data)
{
    /* 1. allocate node */
    Node* new_node = new Node;

    struct Node* last = *head_ref; /* used in step 5*/

    /* 2. put in the data */
    new_node->data = new_data;

    /* 3. This new node is going to be the last node, so
        make next of it as NULL*/
    new_node->next = NULL;

    /* 4. If the Linked List is empty, then make the new
        node as head */
    if (*head_ref == NULL) {
        new_node->prev = NULL;
        *head_ref = new_node;
        return;
    }

    /* 5. Else traverse till the last node */
    while (last->next != NULL)
        last = last->next;
```

```
60
61      /* 6. Change the next of last node */
62      last->next = new_node;
63
64      /* 7. Make last node as previous of new node */
65      new_node->prev = last;
66
67      return;
68  }
69
70  /* Driver program to test above functions*/
71  int main()
72  {
73      /* Start with the empty list */
74      struct Node* head = NULL;
75
76      // Insert 6. So linked list becomes 6->NULL
77      append(&head, 6);
78
79      // Insert 7 at the end.
80      // So linked list becomes 6->7->NULL
81      append(&head, 7);
82
83      // Insert 1 at the end.
84      // So linked list becomes 6->7->1->NULL
85      append(&head, 1);
86
87      // Insert 4 at the end.
88      // So linked list becomes 6->7->1->4->NULL
89      append(&head, 4);
90
91      cout<<"Created DLL is: ";
92      printList(head);
93
94      return 0;
95  }
96
```

Output:

```
Created DLL is:
Traversal in forward direction
6 7 1 4
Traversal in reverse direction
4 1 7 6
```