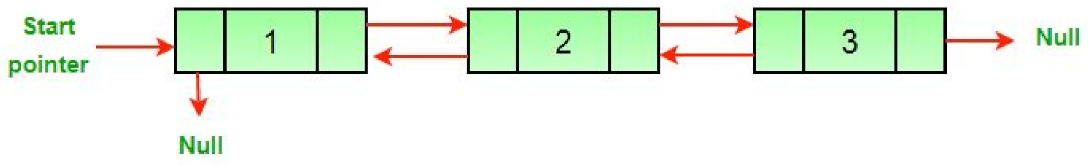
Reverse a Doubly Linked List

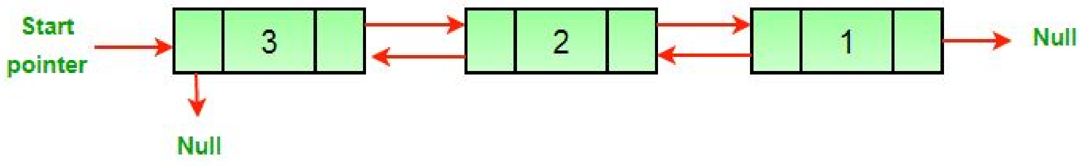
Given a Doubly Linked List, the task is to reverse the given Doubly Linked List.

See below diagrams for example.

```
        (a) Original Doubly Linked List
```



```
     (b) Reversed Doubly Linked List
```



Here is a simple method for reversing a Doubly Linked List. All we need to do is swap prev and next pointers for all nodes, change prev of the head (or start) and change the head pointer in the end.

```
/* C++ program to reverse a doubly linked list */
#include <bits/stdc++.h>
using namespace std;

/* a node of the doubly linked list */
class Node
{
    public:
    int data;
    Node *next;
    Node *prev;
};

/* Function to reverse a Doubly Linked List */
void reverse(Node **head_ref)
{
    Node *temp = NULL;
    Node *current = *head_ref;

    /* swap next and prev for all nodes of
    doubly linked list */
    while (current != NULL)
    {
        temp = current->prev;
        current->prev = current->next;
        current->next = temp;
        current = current->prev;
    }

    /* Before changing the head, check for the cases like emp
        list and list with only one node */
    if(temp != NULL )
        *head_ref = temp->prev;
}



/* UTILITY FUNCTIONS */
/* Function to insert a node at the
beginning of the Doubly Linked List */
void push(Node** head_ref, int new_data)
{
    /* allocate node */
    Node* new_node = new Node();

    /* put in the data */
    new_node->data = new_data;
    /* since we are adding at the beginning,
    prev is always NULL */
    new_node->prev = NULL;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* change prev of head node to new node */
    if((*head_ref) != NULL)
    (*head_ref)->prev = new_node ;

    /* move the head to point to the new node */
    (*head_ref) = new_node;
}

/* Function to print nodes in a given doubly linked list
This function is same as printList() of singly linked list */
void printList(Node *node)
{
    while(node != NULL)
    {
        cout << node->data << " ";
        node = node->next;
    }
}

/* Driver code */
int main()
{
    /* Start with the empty list */
    Node* head = NULL;

    /* Let us create a sorted linked list to test the functio
    Created linked list will be 10->8->4->2 */
    push(&head, 2);
    push(&head, 4);
    push(&head, 8);
    push(&head, 10);

    cout << "Original Linked list" << endl;
    printList(head);

    /* Reverse doubly linked list */
    reverse(&head);

    cout << "\nReversed Linked list" << endl;
    printList(head);

    return 0;
}
```

Output:

```
Original linked list
10 8 4 2
The reversed Linked List is
2 4 8 10
```

Time Complexity: *O(N), where N denotes the number of nodes in the doubly linked list.*

Auxiliary Space: O(1)

We can also swap data instead of pointers to reverse the Doubly Linked List. Method used for reversing array can be used to swap data. Swapping data can be costly compared to pointers if the size of the data item(s) is more.

Please write comments if you find any of the above codes/algorithms incorrect, or find better ways to solve the same problem.

Method 2:

The same question can also be done by using Stacks.

Steps:

1. Keep pushing the node's data in the stack. -> O(n)
2. The keep popping the elements out and updating the Doubly Linked List

```
// C++ program to reverse a doubly linked list
#include <bits/stdc++.h>
using namespace std;
struct LinkedList {
    struct Node {
        int data;
        Node *next, *prev;
        Node(int d)
        {
            data = d;
            next = prev = NULL;
        }
    };
    Node* head = NULL;

    /* Function to reverse a Doubly Linked List using Stacks
     */
    void reverse()
    {
        stack<int> st;
        Node* temp = head;
        while (temp != NULL) {
            st.push(temp->data);
            temp = temp->next;
        }

        // added all the elements sequence wise in the
        // st
        temp = head;
        while (temp != NULL) {
            temp->data = st.top();
            st.pop();
            temp = temp->next;
        }

        // popped all the elements and the added in the
        // linked list,
        // which are in the reversed order->
    }

    /* UTILITY FUNCTIONS */
    /* Function to insert a node at the beginning of the
     * Doubly Linked List */
    void Push(int new_data)
    {

        /* allocate node */
        Node* new_node = new Node(new_data);

        /* since we are adding at the beginning,
         prev is always NULL */
        new_node->prev = NULL;

        /* link the old list off the new node */
        new_node->next = head;

        /* change prev of head node to new node */
        if (head != NULL) {
            head->prev = new_node;
        }

        /* move the head to point to the new node */
        head = new_node;
    }

    /* Function to print nodes in a given doubly linked list
     This function is same as printList() of singly linked
     list */
    void printList(Node* node)
    {
        while (node) {
            cout << node->data << " ";
            node = node->next;
        }
    }
};
```

```
// Driver Code
int main()
{
    LinkedList list;

    /* Let us create a sorted linked list to test the
     functions Created linked list will be 10->8->4->2
   */
    list.Push(2);
    list.Push(4);
    list.Push(8);
    list.Push(10);
    cout << "Original linked list " << endl;
    list.printList(list.head);
    list.reverse();
    cout << endl;
    cout << "The reversed Linked List is " << endl;
    list.printList(list.head);
}
```

Output

```
 Original linked list 
10 8 4 2 
The reversed Linked List is 
2 4 8 10
```

Time Complexity: *O(N)*

Auxiliary Space: *O(N)*

In this method, we traverse the linked list once and add elements to the stack, and again traverse the whole for updating all the elements. The whole takes 2n time, which is the time complexity of O(n).