Reverse a LL in C++

```
#include <iostream>
using namespace std;

// Node class definition
class Node {
public:
    int data;
    Node* next;

    Node(int d) {
        data = d;
        next = nullptr;
    }
};

// Function to display the linked list
void display(Node* head) {
    while (head != nullptr) {
        cout << head->data;
        if (head->next != nullptr) {
            cout << "->";
        }
        head = head->next;
    }
    cout << endl;
}

// Function to reverse the linked list recursively
Node* reverse(Node* head) {
    if (head == nullptr || head->next == nullptr) {
        return head;
    }
    Node* smallAns = reverse(head->next);
    head->next->next = head;
    head->next = nullptr;
    return smallAns;
}

// Function to reverse the linked list iteratively
Node* reverseI(Node* head) {
    if (head == nullptr || head->next == nullptr) {
        return head;
    }
    Node* prev = nullptr;
    Node* curr = head;
    Node* next = nullptr;
    while (curr != nullptr) {
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }
    return prev;
}

int main() {
    // Creating the linked list
    Node* one = new Node(1);
    Node* two = new Node(2);
    Node* three = new Node(3);
    Node* four = new Node(4);
```

Dry Run Table (Step-by-step Iteration)

Iteration	curr->data	next->data	prev->data	What Happens	List State
0	1	2	nullptr	Reverse 1->nullptr, move prev = 1, curr = 2	1
1	2	3	1	Reverse 2->1, move prev = 2, curr = 3	2 -> 1
2	3	4	2	Reverse 3->2, move prev = 3, curr = 4	3 -> 2 -> 1
3	4	5	3	Reverse 4->3, move prev = 4, curr = 5	4 -> 3 -> 2 -> 1
4	5	6	4	Reverse 5->4, move prev = 5, curr = 6	5 -> 4 -> 3 -> 2 -> 1
5	6	7	5	Reverse 6->5, move prev = 6, curr = 7	6 -> 5 -> 4 -> 3 -> 2 -> 1
6	7	nullptr	6	Reverse 7->6, move prev = 7, curr = nullptr	7 -> 6 -> 5 -> 4 -> 3 -> 2 -> 1

✅ **Final Pointers:**

- curr == nullptr → end of list
- prev == 7 → head of reversed list
- So, the function returns prev as the new head.

✅ **Final Output:**

List after iterative reversal: 7->6->5->4->3->2->1

```
    Node* five = new Node(5);
    Node* six = new Node(6);
    Node* seven = new Node(7);
    one->next = two;
    two->next = three;
    three->next = four;
    four->next = five;
    five->next = six;
    six->next = seven;

    // Displaying the original list
    cout << "Original List: ";
    display(one);

    // Reversing the list recursively
    cout << "List after recursive reversal: ";
    Node* revRec = reverse(one);
    display(revRec);

    // Reversing the list iteratively
    cout << "List after iterative reversal: ";
    Node* revIter = reverseI(revRec);
    display(revIter);

    // Deallocating memory
    delete revIter;

    return 0;
}
```

Output:-

List after iterative reversal: 7->6->5->4->3->2->1