Reverse 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);
```

Recursive Reversal: reverse(Node* head)

🔍 Dry Run (for list: 1 -> 2 -> 3)

Step	Call Stack (Function Call)	Action	Resulting Links
1	reverse(1)	Calls reverse(2)	-
2	reverse(2)	Calls reverse(3)	-
3	reverse(3)	Base case hit, returns 3	-
4	Back to reverse(2)	3->next = 2, 2->next = nullptr	3 → 2
5	Back to reverse(1)	2->next = 1, 1->next = nullptr	3 → 2 → 1

✅ Final Result: 3 → 2 → 1

🔁 Iterative Reversal: reverseI(Node* head)

🔍 Dry Run (on 3 → 2 → 1)

curr	prev	next	Action	New Links
3	null	2	3->next = null	3
2	3	1	2->next = 3	2 → 3
1	2	null	1->next = 2	1 → 2 → 3

✅ Final Result: 1 → 2 → 3

```
    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;
}
```

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