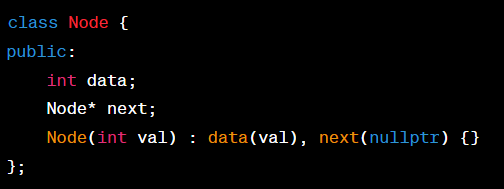
**FINDING INTERSECTION POINT OF TWO SINGLY LINKED LISTS WITH A TWIST**

**Linked List Intersection Challenge**

**Problem**:

You've encountered an interesting linked list challenge that involves finding the intersection point of two singly linked lists. However, there's a twist! You are not allowed to modify the original linked lists by adding any extra data, pointers, or attributes. Your task is to implement a function **Node\* findIntersection(Node\* list1, Node\* list2)** in C++ that returns the intersecting node if it exists, or nullptr if there is no intersection. The catch is that you're only allowed to use constant extra memory (O(1)).

The **Node** class is provided for you and has the following structure:

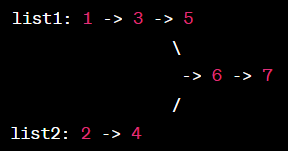


The two input linked lists, **list1** and **list2**, are given as their respective head nodes. It's guaranteed that the lists have a non-negative length and that the intersection point is valid.

**Constraints:**

* You must not modify the original linked lists or introduce any additional data structures.
* Your solution should have a time complexity of O(n) and a space complexity of O(1).

**Example:** Consider the following linked lists:



In this example, the two linked lists intersect at node with value 6. Your function should return a pointer to this node.

**Note:** This challenge is designed to test your ability to work with pointers and linked list traversal efficiently, even under constraints that limit memory usage.