CopyListwithRandomPointers in C++

```
#include <iostream>
#include <unordered_map>

// Definition for a Node.
struct Node {
    int val;
    Node* next;
    Node* random;

    Node(int _val) {
        val = _val;
        next = nullptr;
        random = nullptr;
    }
};

Node* copyRandomList(Node* head) {
    if (head == nullptr) return nullptr;

    std::unordered_map<Node*, Node*> map;
    Node* curr = head;

    // First pass: create all nodes and store them in the
map.
    while (curr != nullptr) {
        map[curr] = new Node(curr->val);
        curr = curr->next;
    }

    // Second pass: assign next and random pointers.
    curr = head;
    while (curr != nullptr) {
        map[curr]->next = map[curr->next];
        map[curr]->random = map[curr->random];
        curr = curr->next;
    }

    return map[head];
}

void printList(Node* head) {
    while (head != nullptr) {
        std::cout << "Node(" << head->val << ")";
        if (head->random != nullptr) {
            std::cout << " [Random(" << head->random-
>val << ")]";
        }
        std::cout << " -> ";
        head = head->next;
    }
    std::cout << "null" << std::endl;
}

int main() {
    Node* head = new Node(1);
    head->next = new Node(2);
    head->next->next = new Node(3);
    head->random = head->next->next;
    head->next->random = head;

    Node* result = copyRandomList(head);
```

Goal: Deep copy a linked list where each node has next and random pointers.

Given input:

```
1 -> 2 -> 3
|    |
v    v
3    1
```

📌 Step-by-Step Dry Run Table

Step	Operation	Affected Node	Explanation
First Pass	map[1] = new Node(1)	Node 1	Creates a copy of node 1
	map[2] = new Node(2)	Node 2	Creates a copy of node 2
	map[3] = new Node(3)	Node 3	Creates a copy of node 3
Second Pass	map[1]->next = map[2]	Node 1 copy	Sets next of copied 1 to copied 2
	map[1]->random = map[3]	Node 1 copy	Sets random of copied 1 to copied 3 (like original)
	map[2]->next = map[3]	Node 2 copy	Sets next of copied 2 to copied 3
	map[2]->random = map[1]	Node 2 copy	Sets random of copied 2 to copied 1
	map[3]->next = map[nullptr] = null	Node 3 copy	Last node, next is null
	map[3]->random = map[nullptr]	Node 3 copy	random was not set originally, stays null

✅ Final Output:

Copied list:

1 [Random(3)] -> 2 [Random(1)] -> 3 -> null

```
    printList(result);

    // Free the allocated memory
    Node* curr = result;
    while (curr != nullptr) {
        Node* temp = curr;
        curr = curr->next;
        delete temp;
    }

    return 0;
}
```

Output:-
0