Multiply LL in C++

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

// Node class for the
linked list
class Node {
public:
    int val;
    Node* next;

    Node(int val) {
        this->val = val;
        this->next = nullptr;
    }
};
Node* reverse(Node*
head) {
    if (head == nullptr ||
head->next == nullptr)
return head;

    Node* prev = nullptr;
    Node* curr = head;
    while (curr != nullptr) {
        Node* forw = curr-
>next;
        curr->next = prev;
        prev = curr;
        curr = forw;
    }

    return prev;
}

// Function to add two
linked lists in place
void
addTwoLinkedList(Node*
head, Node* ansItr) {
    Node* c1 = head;
    Node* c2 = ansItr;

    int carry = 0;
    while (c1 != nullptr ||
carry != 0) {
        int sum = carry + (c1
!= nullptr ? c1->val : 0) +
(c2->next != nullptr ? c2-
>next->val : 0);
        int digit = sum % 10;
        carry = sum / 10;

        if (c2->next !=
nullptr) c2->next->val =
digit;
        else c2->next = new
Node(digit);

        if (c1 != nullptr) c1 =
c1->next;
        c2 = c2->next;
```

Given:

- **l1 = 2 -> 4 -> 3** (representing the number 342)
- **l2 = 5 -> 6 -> 4** (representing the number 465)

We are multiplying these two numbers, and as part of the algorithm, we reverse both linked lists, perform multiplication on each digit, and handle carries. Then, we add the intermediate results, ensuring proper shifting of digits.

Dry Run Table:

Step	l1 (reversed)	l2 (reversed)	Current digit of l2 (l2_itr->val)	Multiplication Result (prod)	Shift Applied	Interim Result
Initial	3 -> 4 -> 2	4 -> 6 -> 5	N/A	N/A	N/A	N/A
Reversed	2 -> 4 -> 3	5 -> 6 -> 4	N/A	N/A	N/A	N/A
Multiply l1 by 5 (1st digit of l2)	2 -> 4 -> 3	5	5 * 3 = 15, 5 * 4 = 20 + 1 (carry) = 21, 5 * 2 = 10 + 2 (carry) = 12	5 -> 1 -> 2	No Shift (first digit)	5 -> 1 -> 2
Add this result to the intermediate result (result = 5 -> 1 -> 2)	2 -> 4 -> 3	6 -> 5	N/A	N/A	N/A	5 -> 1 -> 2 (no change)
Multiply l1 by 6 (2nd digit of l2)	2 -> 4 -> 3	6	6 * 3 = 18, 6 * 4 = 24 + 1 (carry) = 25, 6 * 2 = 12 + 2 (carry) = 14	8 -> 5 -> 4	Shift by 1	8 -> 5 -> 4 -> 0 -> 0
Add this result to the intermediate result (add 8 -> 5 -> 4 -> 0 -> 0 to 5 -> 1 -> 2)	2 -> 4 -> 3	5	N/A	N/A	N/A	1 -> 5 -> 9 -> 0 -> 3 -> 0
Multiply l1 by 4 (3rd digit of l2)	2 -> 4 -> 3	4	4 * 3 = 12, 4 * 4 = 16 + 1 (carry) = 17, 4 * 2 = 8 + 1 (carry) = 9	2 -> 7 -> 9	Shift by 2	2 -> 7 -> 9 -> 0 -> 0 -> 0
Add this result to the intermediate result (add 2	2 -> 4 -> 3	4	N/A	N/A	N/A	1 -> 5 -> 9 -> 0 -> 3 -> 0 (final

```
    }
}

// Function to multiply a linked list with a single digit
Node* multiplyLLWithDigit(Node* head, int dig) {
    Node* dummy = new Node(-1);
    Node* ac = dummy;
    Node* curr = head;
    int carry = 0;
    while (curr != nullptr || carry != 0) {
        int sum = carry + (curr != nullptr ? curr->val * dig : 0);

        int digit = sum % 10;
        carry = sum / 10;

        ac->next = new Node(digit);

        if (curr != nullptr) curr = curr->next;
        ac = ac->next;
    }

    return dummy->next;
}

// Function to multiply two linked lists representing numbers
Node* multiplyTwoLL(Node* l1, Node* l2) {
    l1 = reverse(l1);
    l2 = reverse(l2);

    Node* l2_Itr = l2;
    Node* dummy = new Node(-1);
    Node* ansItr = dummy;

    while (l2_Itr != nullptr) {
        Node* prod = multiplyLLWithDigit(l1, l2_Itr->val);
        l2_Itr = l2_Itr->next;


addTwoLinkedList(prod, ansItr);
        ansItr = ansItr->next;
    }
```

-> 7 -> 9 -> 0 -> 0 -> 0 to 1 -> 5 -> 9 -> 0 -> 3 -> 0)						result)

Step-by-Step Process:

1. **Reversing the Lists**:
   - l1 = 2 -> 4 -> 3 becomes 3 -> 4 -> 2.
   - l2 = 5 -> 6 -> 4 becomes 4 -> 6 -> 5.
2. **Multiplying l1 by each digit of l2**:
   - **First, multiply l1 by 5**:
     - 5 * 3 = 15, carry = 1.
     - 5 * 4 = 20 + 1 (carry) = 21, carry = 2.
     - 5 * 2 = 10 + 2 (carry) = 12, carry = 1.
     - Result: 5 -> 1 -> 2.
   - **Second, multiply l1 by 6** (shifting by one place):
     - 6 * 3 = 18, carry = 1.
     - 6 * 4 = 24 + 1 (carry) = 25, carry = 2.
     - 6 * 2 = 12 + 2 (carry) = 14, carry = 1.
     - Result: 8 -> 5 -> 4 -> 0 -> 0.
   - **Third, multiply l1 by 4** (shifting by two places):
     - 4 * 3 = 12, carry = 1.
     - 4 * 4 = 16 + 1 (carry) = 17, carry = 1.
     - 4 * 2 = 8 + 1 (carry) = 9, carry = 0.
     - Result: 2 -> 7 -> 9 -> 0 -> 0.
3. **Adding the Intermediate Results**:
   - Add the first product 5 -> 1 -> 2 to the result.
   - Add the second product 8 -> 5 -> 4 -> 0 -> 0 to the result.
   - Add the third product 2 -> 7 -> 9 -> 0 -> 0 -> 0 to the result.
4. **Final Output**:
   - The result after adding all the intermediate products is 1 -> 5 -> 9 -> 0 -> 3 -> 0, which is the correct result for 342 * 465 = 159030.

Final Output:

159030

```
    return reverse(dummy->next);
}

// Function to print the linked list
void printList(Node* node) {
    while (node != nullptr) {
        cout << node->val << " ";
        node = node->next;
    }
    cout << endl;
}

// Function to create a linked list from an array of integers
Node* createList(int values[], int n) {
    Node* dummy = new Node(-1);
    Node* prev = dummy;
    for (int i = 0; i < n; ++i) {
        prev->next = new Node(values[i]);
        prev = prev->next;
    }
    return dummy->next;
}

int main() {
    // Hardcoding the lists
    // First list: 3 -> 4 -> 2 (represents the number 243)
    int arr1[] = {3, 4, 2};
    int n1 = sizeof(arr1) / sizeof(arr1[0]);
    Node* head1 = createList(arr1, n1);

    // Second list: 4 -> 6 -> 5 (represents the number 564)
    int arr2[] = {4, 6, 5};
    int n2 = sizeof(arr2) / sizeof(arr2[0]);
    Node* head2 = createList(arr2, n2);

    // Multiplying the two linked lists
    Node* ans = multiplyTwoLL(head1, head2);

    // Printing the result
```

printList(ans);
 	
return 0;
}	

1 5 9 0 3 0	