4. Braiding a Linked List (`braid.cpp`)

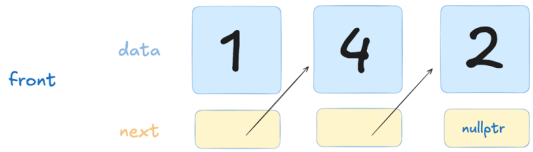
Write a function braid that takes a linked list and weaves the reverse of that list into the original. (You will need to create new nodes.) Here are a few examples:

```
{1, 4, 2} -> {1, 2, 4, 4, 2, 1}
{3} -> {3, 3}
{1, 3, 6, 10, 15} -> {1, 15, 3, 10, 6, 6, 10, 3, 15, 1}
```

You should implement your code to match the following prototype

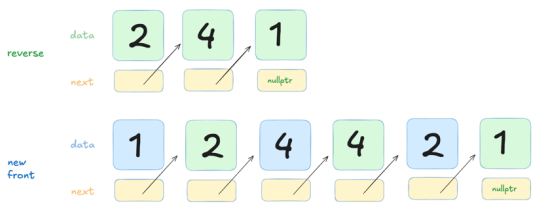
```
void braid(Node*& front)
```

We are passed in a linked list by reference, front.




```
struct Node {
    int data;
    Node *next;
};
```

We want to *braid* the linked list by weaving the reverse of that list into the original.

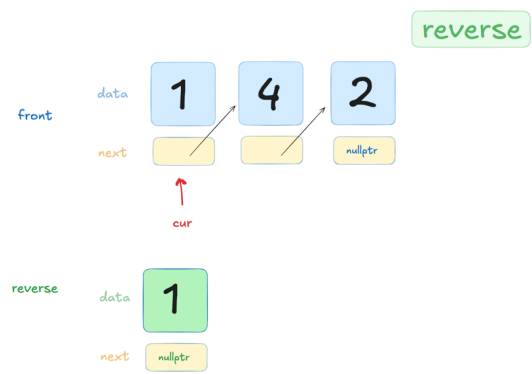


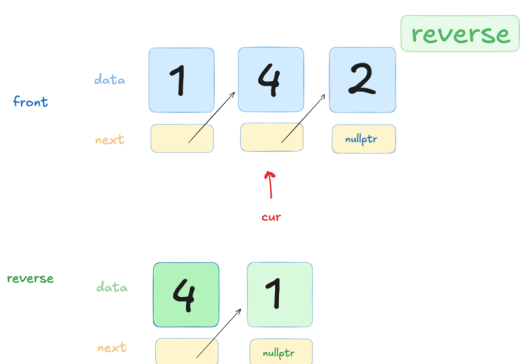
To do this, we break the problem into two steps.

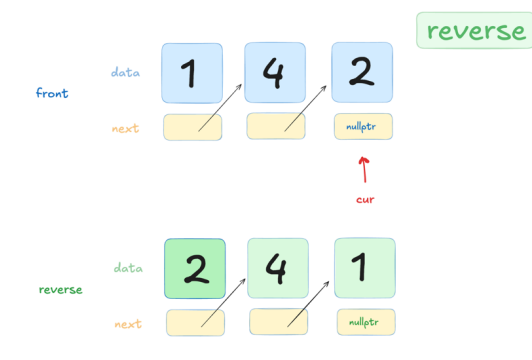
1) We reverse the linked list.
2) We braid the reversed list back into front.

reverse

To reverse a linked list, we can loop through the original list using a pointer cur and headInsert a node containing cur->data into a new list.







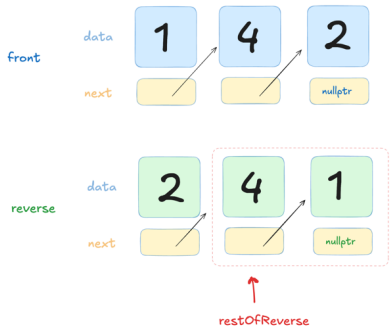
braid

We then want to braid the new reverse list into front.

Note: the code for each line is in blue!

```
Node *restOfReverse = reverse->next;
```

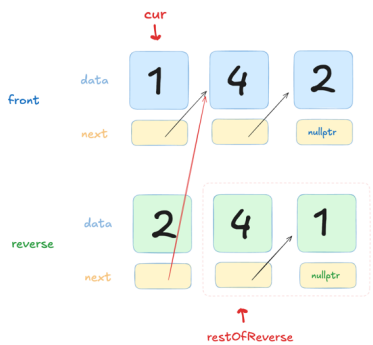
braid



First, we add a pointer to reverse->next so that the nodes in the red dotted box do not get stranded.

```
reverse->next = cur->next;
```

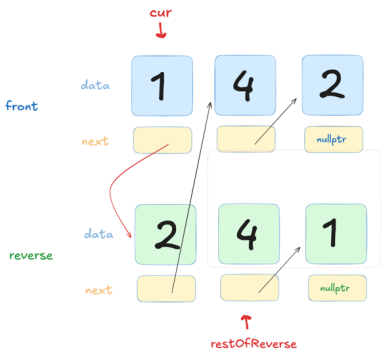
braid



Then, we rewire reverse->next to cur->next.

```
cur->next = reverse;
```

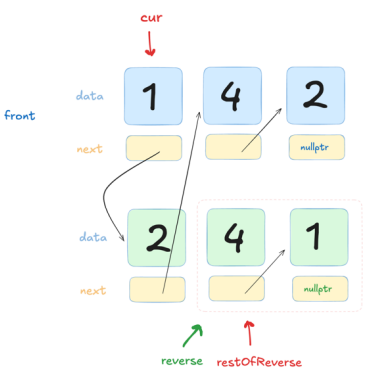
braid



Next, we rewire cur->next to reverse.

```
reverse = restOfReverse;
```

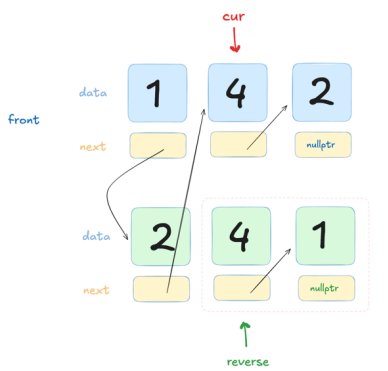
braid



Now that the two has been braided in, we set reverse to restOfReverse.

```
cur = cur->next->next;
```

braid



We move cur forward, and we keep on doing this until we finish traversing the list with cur.

*note that we move cur to cur->next->next here, because cur->next would be green element from the reverse list that we just braided in.

All done!!!

```
void braid(Node*& front) {
    Node *reverse = nullptr;
    for (Node *curr = front; curr != nullptr; curr = curr->next) {
        Node *newNode = new Node;
        newNode->data = curr->data;
        newNode->next = reverse;
        reverse = newNode;
    }
    // reverse now addresses a memory-independent copy of the original list,
    // where all of the nodes are in reverse order.
    for (Node *curr = front; curr != nullptr; curr = curr->next->next) {
        Node *next = reverse->next;
        reverse->next = curr->next;
        curr->next = reverse;
        reverse = next;
    }
}
```