Recursive Implementation of the `removeAll()` function of the Lab 10

In order to implement a function recursively, first you need to design a divide-and-conquer solution to your problem. To do so, as explained in class, first start with the base case(s) of your solution: All cases you can produce answer with no recursive call to the function. For the `removeAll()` function, the base case is related to the case that the list is empty: simply return from the function.

Next, you need to find a recursive solution for the cases which are not included in any of the base cases. At this step you need to do one or more recursive calls on the function by passing proper arguments creating a smaller problem. For example, for the `removeAll()` function, this can be done as follows:

If required remove the first node of the current list, and call the `removeAll()` function (recursively) to remove all the occurrences of the element in a list containing all the elements but the first one.

This algorithm has been implemented below:

```
void LinkedList::removeAll(int value) {
      recursiveRemoveAll(value, head);
}

void LinkedList::recursiveRemoveAll(int value, Node* &list) {
      if (list == nullptr)
            return;
      else if (list->element == value) {
            list = list->next;
            recursiveRemoveAll(value, list);
      }
      else
            recursiveRemoveAll(value, list -> next);

}
```

Please notice that here an integer linked list has been assumed for the sake of simplicity. As you see in the above code, **we do not require to keep track of the previous node**, as it is usually done in the iterative implementation. Actually, passing the list pointer by reference guarantees that all the pointers are set correctly, and we won't have a disconnected list or missed nodes at all.