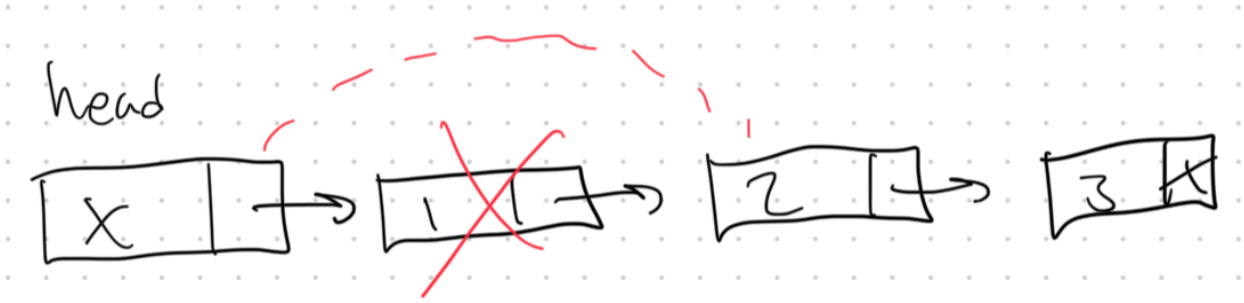
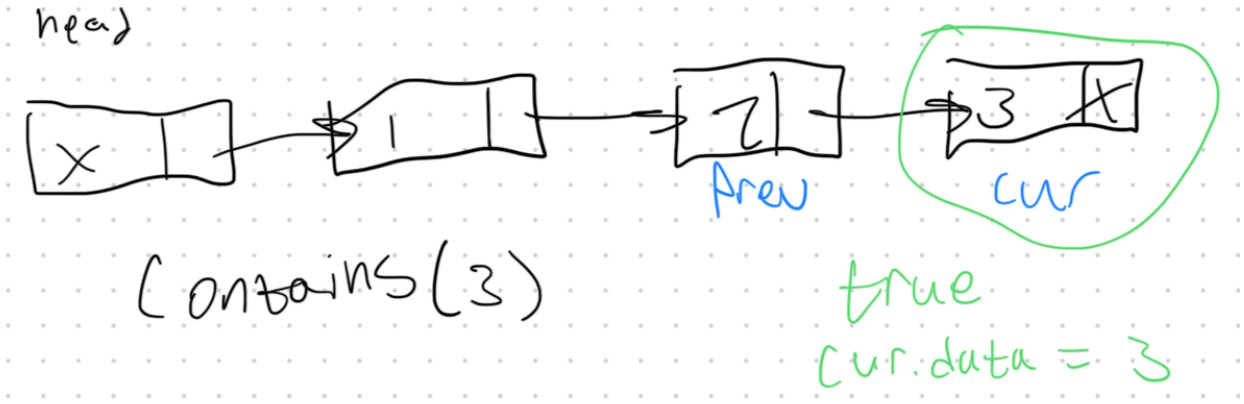
Homework 1 Logic

removeFirst()



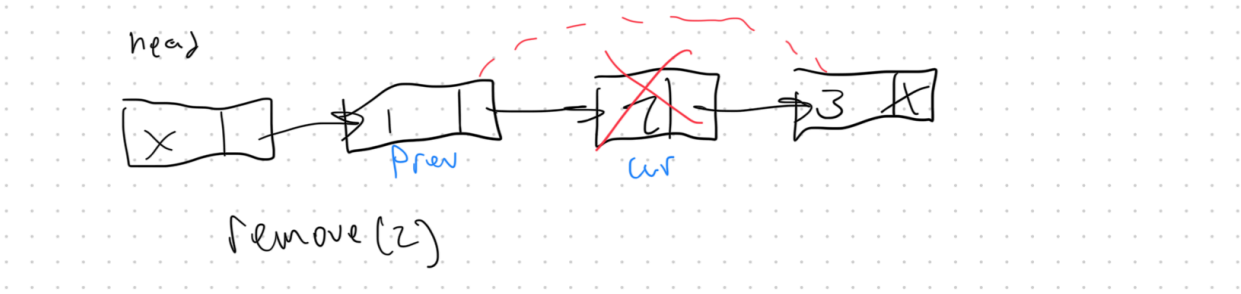
1. Check if the list is empty, if so there is nothing to remove so thrown an exception
2. Store the data form the node we are going to remove
3. Point our dummy head's next to the node after the next node
4. Decrease the size
5. Return our stored data from step 2

contains(Object o)



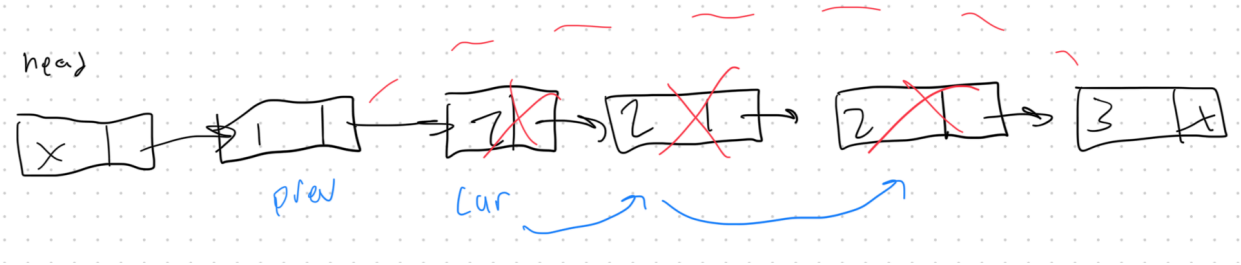
1. Check if the list is empty, if so we return false because nothing can contain Object o
2. Check if the first node and the element being compared is null, if so we return true because our target object was found
3. Iterate through the list while checking if the target object is equal to the current node's data. Return true if we find the target, return false if we don't
4. After the entire iteration is over we can return false because that would mean we went through the whole list without seeing our target object

remove(Object o)



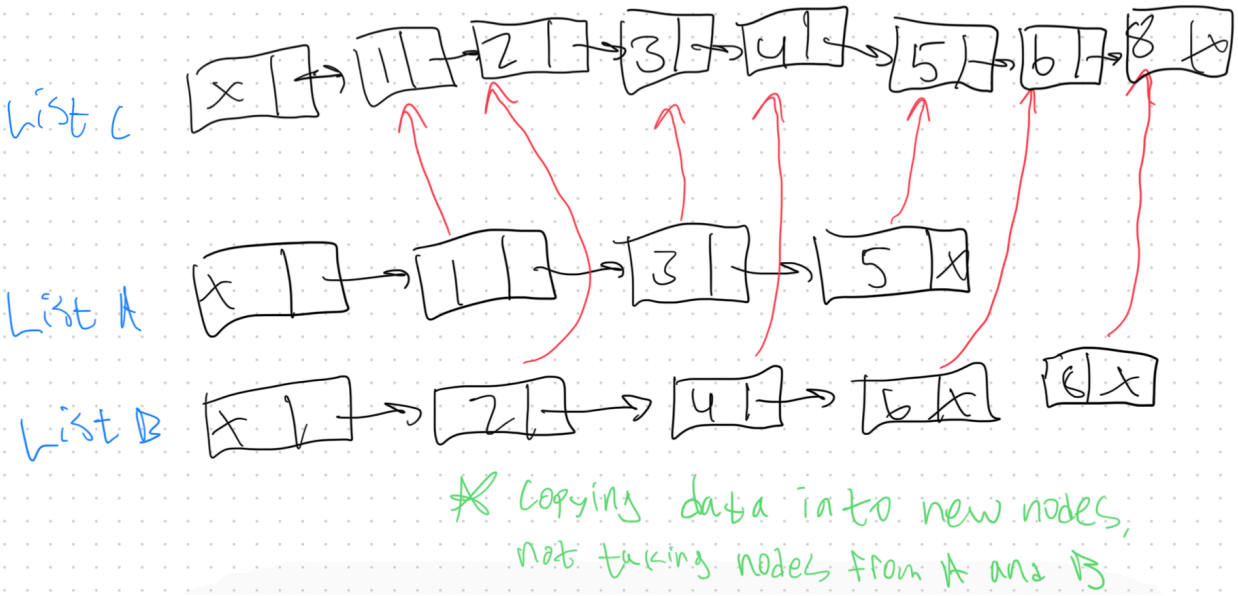
1. Check if the list is empty, if so there is nothing to remove so thrown an exception
2. Iterate through the list with the extra condition that the current node's data is not equal to the target data, so we don't skip over deleting the target data
3. Once the current node is either our target data or our current node is null, check if the current node is null.
4. If the current node is not null, that means we are at our target data. We set the previous node to point to the current node's next node
5. Decrease the size and return true
6. In the case that our current node is null, return false because that means we reached the end of our list without finding a node with our target data

removeAllCopies(Object o)



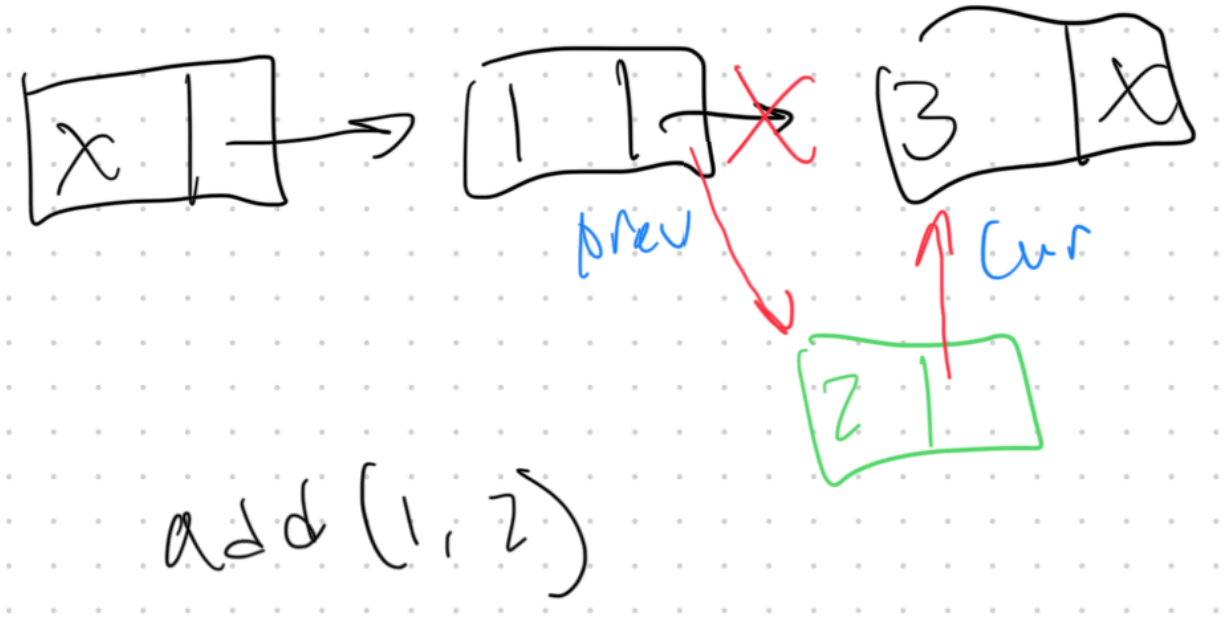
1. Set a boolean value removed to be false, because we have not removed anything yet. Check if the list is empty, if so there is nothing to remove so thrown an exception
2. Iterate through the whole list while checking for conditions
3. Check if the current node's data is our target data. If it is, we set the previous node to point to the current node's next node, we then sat the current node to the previous node's next node, and decrease the size
4. At the end of our first if statement we set our boolean value removed to be true because we have removed at least one node with our target data from the list
5. In the case that our current node's data does not equal our target data, we step through the list
6. We return our boolean value removed

interleave(MyLinkedList A, MyLinkedList B)



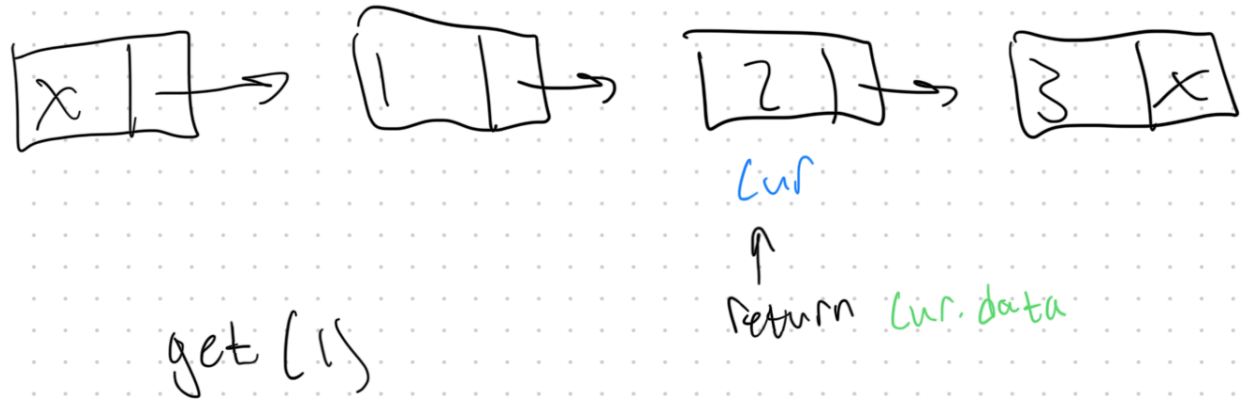
1. Create a new empty list C
2. Iterate completely through both list A and list B
3. While we iterate, we are adding a new node to C with the data from A's current node
4. We increment both C and A's current nodes
5. We then repeat the process for adding a new node with B's current data. This ensures that we are alternating the nodes between A and B without changing their structure or data
6. After we have either gone completely through A or B, check to see if there are still nodes to go through in either list
7. If there are, we append the remaining nodes from whichever list was longer to list C
8. After all of this, we return list C

add(int index, Object o)



1. First check if the index passed in is less that zero or greater than the list's size
2. Throw an exception if either of these conditions are met
3. Walk through the list until we get to the target index
4. Create a new node with the data that will be passed in
5. Set the previous node's next to be the new node
6. Set the new node's next to be the current node
7. Increase the size

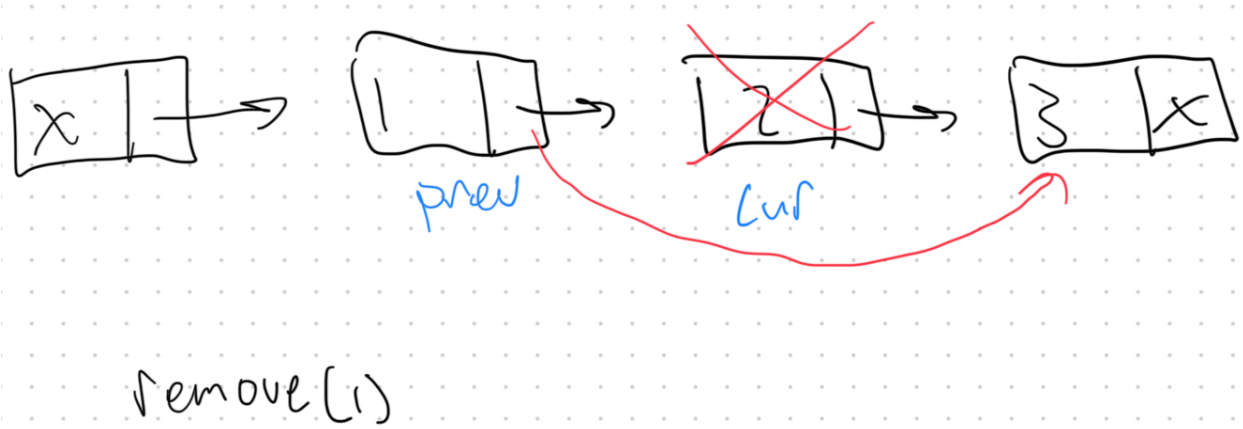
get(int index)



1. Check if the index passed in is less than 0 or equal to or greater than our list's size. If it is we throw an exception
2. Check if the index is equal to the size of the list
3. If it is, we traverse the entire list and return the last node with data
4. Otherwise, we traverse the list until our current node is at the target index

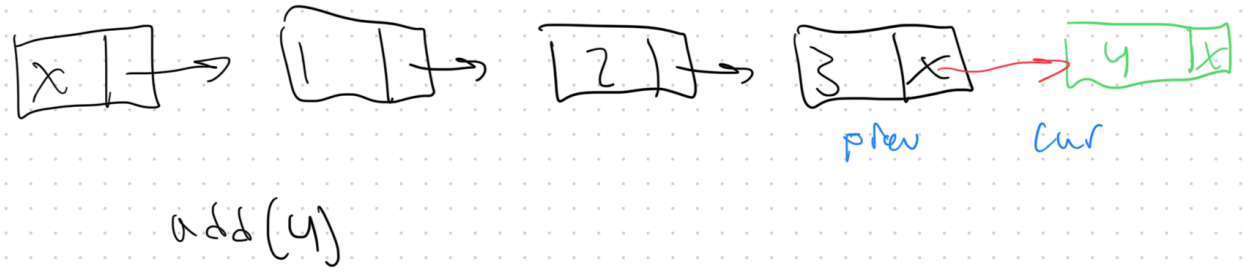
5. The current node is now at the target index so we return the current node's data

remove(int index)



1. Check if the index passed in is less than 0 or equal to or greater than our list's size. If it is we throw an exception
2. Otherwise we traverse the list until our current node is at our target index
3. Once our current node is at our target index, we create a temp object to store our removed data
4. We set our previous node's next node to be our current node's next node
5. Decrease the size
6. Return our removed data

add(Object e)



1. Traverse our list until we are at the very end
2. Set our previous node's next node to be a new node with our passed in data