# Linked List Deletion

This worksheet is designed to help you plan out your deletion code before you start typing.

## The Default Case

The “ordinary” case for deletion is when you are deleting a node from the interior of a list (i.e. neither the first node, nor the last node), when the list has more than a single node in it. This is the “find previous and swoop” method discussed in lecture. Example code for this case can be found on the lecture PowerPoint.

## Special Cases

There are three special cases for deletion: deleting the first node when there is more than one node in the list, deleting the only node from a single node list, and deleting the final node when there is more than one node in the list. In your code you must deal with each of these cases. To figure out how to do this, you need to understand exactly how the list is changed in each case.

Complete this table by drawing the “after” sketches. Be sure to make Head and Tail correct so as to maintain the integrity of the list.

|  |  |  |
| --- | --- | --- |
| **Special Case** | **List Before Deletion** | **List After Deletion** |
| Deleting the first node when there are multiple nodes in the list | H T  nodeToDelete |  |
| Deleting the only node from a list with only one node | H T  nodeToDelete |  |
| Deleting the last node in the list when there are multiple nodes in the list | H T  nodeToDelete |  |

The first two cases make sort of a pair, since they both occur when you are deleting the node pointed to by Head. Since in both cases there is no “previous” node, there is no traversing or swooping. You simply update the Head and Tail pointers as appropriate to make the list look as it should after the first node is deleted.

The third case is different, because you can use your “find previous and swoop” code. It is only ***after*** this code has been executed that you must make a special adjustment to keep Tail pointing where it should.

Thus, the pseudocode for the complete delete method is:

If you are deleting the first node

{

If it is the only node in the list

Head and Tail both become nullptr

else

Move Head along to the second node

}

else

{

Find previous and swoop (see the lecture PowerPoint for discussion)

If you have just deleted the last node in the list

Set Tail to point to the “previous” node, because it is now at the end

}

Spend some time thinking about those if-statements. How can you tell that you are trying to delete the first node in the list? (Recall that a pointer to the node to be deleted is always passed into the delete method.) How can you tell if there is only one node in the list? How can you tell if you are deleting the final node in the list? All of these conditions can be detected by using Head and Tail, which point to the first and last nodes of the list, respectively.

Deletion is the hardest part of the linked list implementation. Luckily, the method is the same for all linked lists you will use during this paper. You only need to write it once, then you can use it over and over.

Good luck with this, and if you get stuck, please let me know.