This is the fourth lecture on linked list.

This lecture talks about the delete function. This function deletes only one node in a linked list. This is different from the destroy function described in the third lecture. The destroy function deletes every node.

The delete function has to consider three different cases. The first case is when the linked list is empty. There is nothing in the linked list.

In the second case, the node to be deleted is the very first node of the list. In other words, this is the head of the list. We need to save the second node, free the first node, and return the second node. The second node becomes the first node after the delete function.

In the third case, the node to be deleted is not the first node. Then, we need to find the node and the node in front of it. The link needs to bypass the node to be deleted. Free the node. Finally, return the original head of the linked list.

Let’s read the code that implement the delete function.

This is the delete function. It takes two arguments. The first is a pointer to the first node of the linked list. The second is a value. This function returns the address of the new linked list after deleted a node whose value matches is the second argument.

If h’s value is NULL, the input linked list is empty and nothing can be deleted. If the value is not stored in the linked list, the linked list should be unchanged. If several nodes store the same value, this function deletes the first node.

We have seen this many times. The beginning of the function checks whether h is NULL or not. If h is NULL, the list is empty and nothing can be deleted. The function returns NULL.

If H is not NULL, we consider the second case. This case is when the first node is the node to delete.

We need to check whether the node’s value is the same as the second argument. If they match, the first node will be deleted.

Before deleting the first node, we need to save the next node. The next node is stored in the pointer P. . It is possible that P is NULL because this linked list has only one node. This is OK.

The memory for the first node is freed and the function returns P. .

Next, we consider the third case. This case is more complex.

Let’s take a quick look of the program and we will go through the program line by line.

We need to have two more pointers, P and Q. .

Q will be the node to be deleted. P is the node before Q. .

Suppose we want to delete the node whose value is 68.

We need to first find the node whose value is 68.

The program has to find the node that will be deleted. If Q is not NULL and Q’s value does not match, P moves to the next node.

Q also moves to the next node.

Please notice that moving both P and Q will keep P as the node in front of Q. .

The program goes back to check whether Q is NULL. If Q is not NULL, the program checks whether Q’s value matches the value we are looking for.

This time, Q’s value is 68. The two values match.

There are two reasons why the program gets out of the first while. . The first condition is when Q is NULL. This occurs when the value vee is not in the linked list.

The second condition is when Q’s value matches vee.

We need to check whether Q is NULL or not. If it is not NULL, then P’s next is set to Q’s next.

What does this mean? It means P’s next bypasses Q and points to the node after Q. .

The next line frees Q. . This node is no longer in the heap memory.

The function returns the first node of the linked list.

This slide reviews the steps for the delete function. We have the check the three cases. The first is when the list is empty. The second case is when the first node is deleted. The third case is when the deleted node is not the first node.

Let me answer some frequently asked questions. The first question is whether we can reorder these three statements. The answer is no. We must store the second node before freeing H. . After freeing H, H’s next does not exist any more.

If we move return P earlier, the function returns to the caller. Thus, the order of these three statements must not change.

The next question is whether we need three pointers: H, P, and Q. .

The answer is Yes. We need H as the first node of the linked list. Without H, we lose the linked list. We need Q as the node to be deleted. We also need P as the node before Q so that the link from P can bypass Q. .

We have seen the next question before. Can we swap the order of these two conditions? The answer is No. We must check whether Q is NULL first. If Q is NULL, the second condition is not checked. If Q is not NULL, the second condition checks whether Q’s value matches vee.

The next question is whether we can reorder these two statements. The answer is Yes. We can move Q first and then move P. .

The next question is whether we can reorder these two statements. The answer is No. After freeing Q, Q’s next does not exist any more.