This is the second lecture on linked list. This lecture focuses on the search function.

What does the search function do? It takes two arguments: the head of the linked list and a value to be searched.

If this value is stored in the linked list, this function returns a pointer to the node that stores the value. If this value is not stored in the linked list, this function returns NULL. If several nodes stores this value, this function returns the first one.

This slide includes information how to call the search function.

Suppose in the caller, the pointer head points to the first node of the linked list. The linked list stores numbers 917, negative 37, 68, and 52. We want to know whether number 68 is stored in the list by calling the search function. It is important that you use a variable different from head. The reason is that if you use head on the left side of the assignment, you will lose the linked list.

Consider this example calling the search function. At the beginning, the argument H stores the address of the first node, 20000.

The function creates a local pointer called P. and its value is the same as H. . That means P points to the same place as H points to.

The function has to check whether P points to a valid piece of memory.

Almost every function related to linked lists have something like this: testing whether a pointer stores NULL or not. This the reason why you must ensure that pointers are correctly initialized.

Inside while, the function compares the value with the input argument Vee. If they are the same, we have found a node that stores the value. The function returns P. .

If the values are different, the function continues to the next line. This line moves P to point to the next node.

As the moment, P points to the second node. P’s value stores the address 50000 and this node stores the value negative 37.

The program goes to check whether P is NULL or not. This is important. If P stores NULL, then there is no node and we must not intend to check the value.

Since P does not store NULL, we are allowed to check whether the value is the same as vee or not. The node’s value is negative 37 and vee is 68. They do not match.

The function goes to the next line. This line moves P to point to the next node.

The function goes back to the top checking whether P is NULL or not.

P’s value is 40000, not NULL. Thus, the program continues to the if condition.

At this moment, the node’s value matches the value of vee. Thus, the search function returns p.

The value of p is returned and saved to the value of Q. .

This is the stack after the research function returns. The top frame has been popped. Please notice that head’s value is still 20000 and it points to the first node of the linked list.

Let’s go over the program again and this time we explicitly show the heap memory.

This slide draws the linked list and the heap memory together before calling the search function.

The variable head stores the address of the first node and the value is 20000.

Each node has two attributes: a pointer called next and an integer called value.

The integer stored at the first node is 917. The address is 20008 because a pointer takes 8 bytes of space.

The next pointer stores address 50000. This is the second node in the linked list. This node’s value is negative 317. The next pointer stores the address 40000. At this third node, the integer is 68 and the next pointer is 30000.

At the beginning of the search function, the pointer P is set to the value 20000. This is the first node in the linked list.

The function checks whether p’s value is NULL or not. Since P’s value is 20000, it is not NULL.

The next step compares this node’s value and vee. This node’s value is 917. Vee is 68. They do not match.

P’s value becomes 50000. The program goes back to check whether P is NULL or not.

Here are a few commonly asked questions. The first question is whether we have to create a local variable p. Can we use h here?

The answer is: We do not need to create a local variable P. . We can use H inside this function.

If we h inside the search function, we will lose the first node of the linked list but this is all right. Outside the search function, the value of head is not changed. Thus, we can still go to the first node of the linked list.

In contrast, outside the search function, we use another variable Q to store the return value of search. Do we need to create the variable Q here? The answer is Yes. Can we store the return value of search in head? The answer is No. We must not modify head. If we modify head, we will lose the first node of the linked list.

We can rewrite the search function in this way. The condition for while can be written this way.

P moves to the next if both conditions are satisfied. First, P is not NULL. Second, the value is not vee.

If either condition is false, P will not move.

Let’s carefully examine how this works.

In a C program, when a condition has two parts: ei and B, then the program checks ei first. If ei is false, the condition is false and the program does not check B.

For the condition in while, if P is NULL, the condition is false. The program returns P. This means the value is not found.

If P is not NULL, the first of the condition is true. Thus, the program continues checking the second part of the condition. If the value is different from vee, P moves to the next. If the value is the same, while stops without moving P. .

The returned value of P is the pointer to the node that stores the value of vee.

It is important to understand that the order of the two conditions is important. We must not check the value before we check whether p is NULL.

If P is NULL, P arrow value does not exist and this program will have segmentation fault.

Thus, we have to check whether P is NULL first.