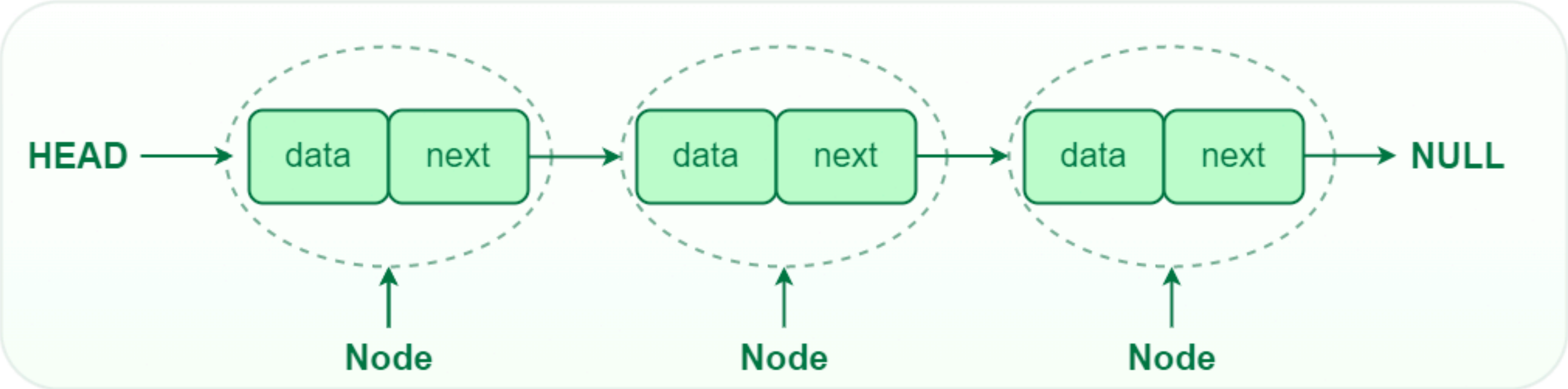
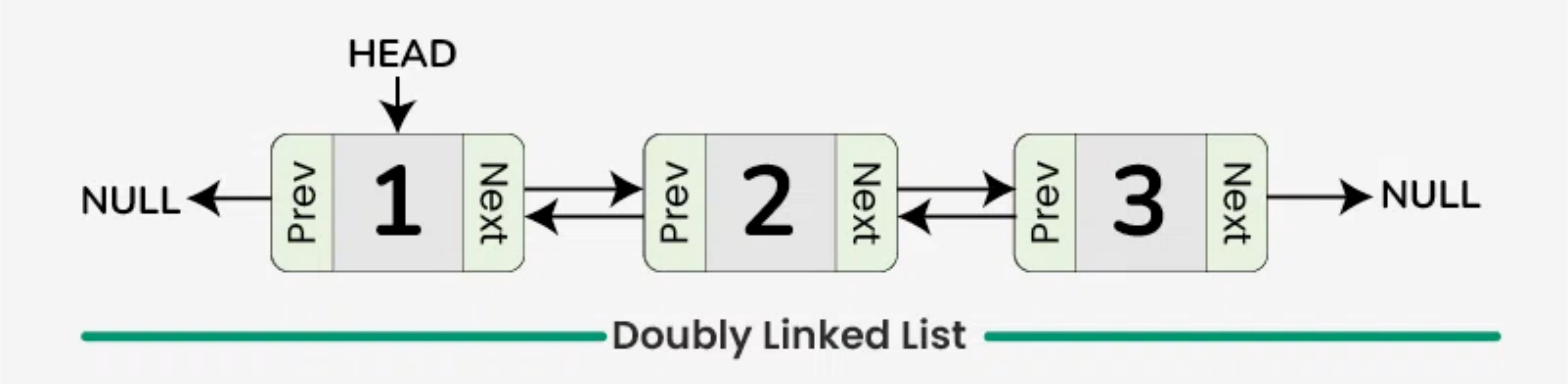
CS62 Class 8: Linked lists

Basic Data Structures



https://www.geeksforgeeks.org/introduction-to-singly-linked-list/



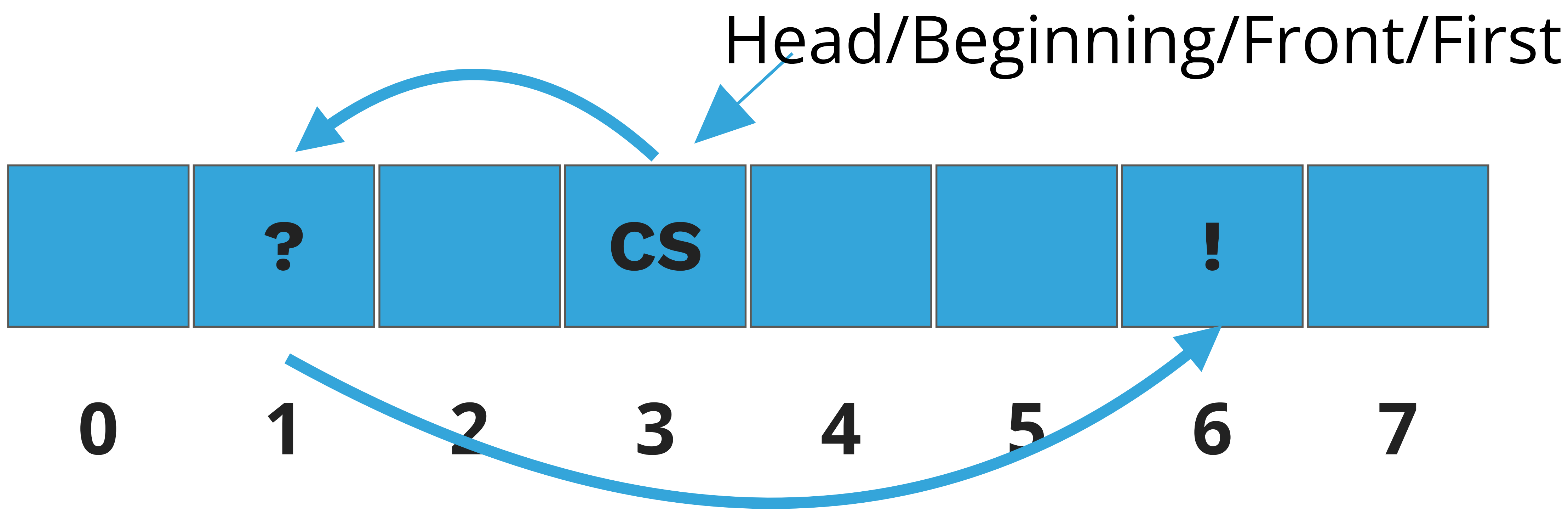
https://www.geeksforgeeks.org/doubly-linked-list/

Linked Lists (conceptually)

Linked Lists

- Dynamic linear data structures.
- In contrast to sequential data structures, linked data structures use pointers/links/ references from one object to another.
    - For example, the list of elements CS, ?, ! could be in very different memory locations. We just need a pointer to the head and links to subsequent elements to reconstruct it.

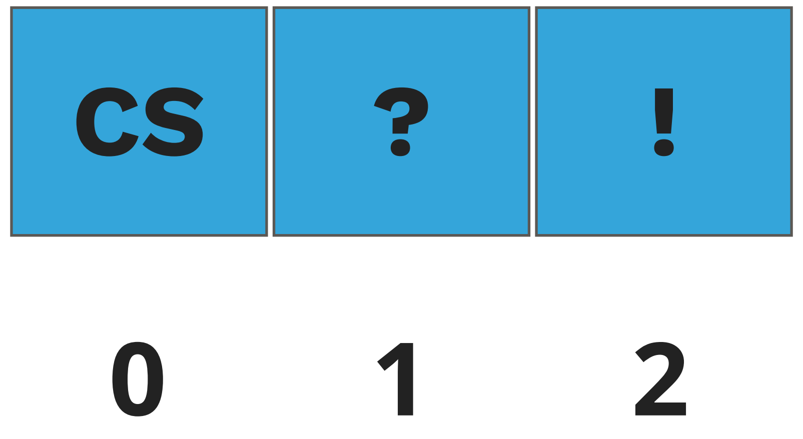
How it is in memory:



Head/Beginning/Front/First

	?		CS			!	
0	1	2	3	4	5	6	7

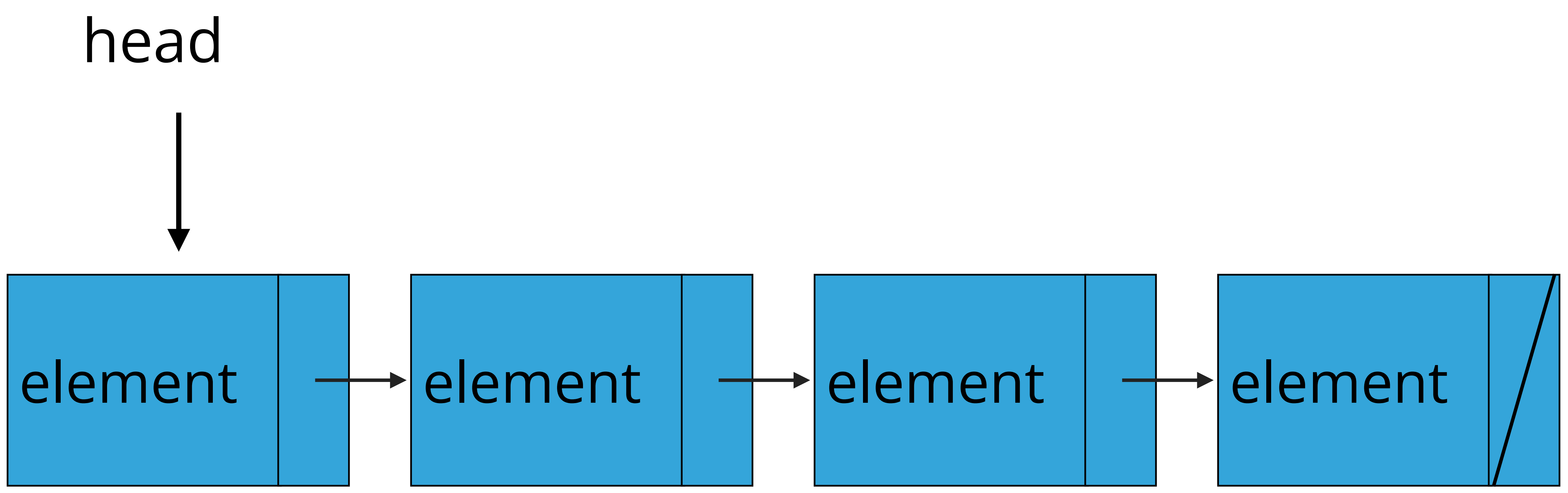
What the user’s mental model is:

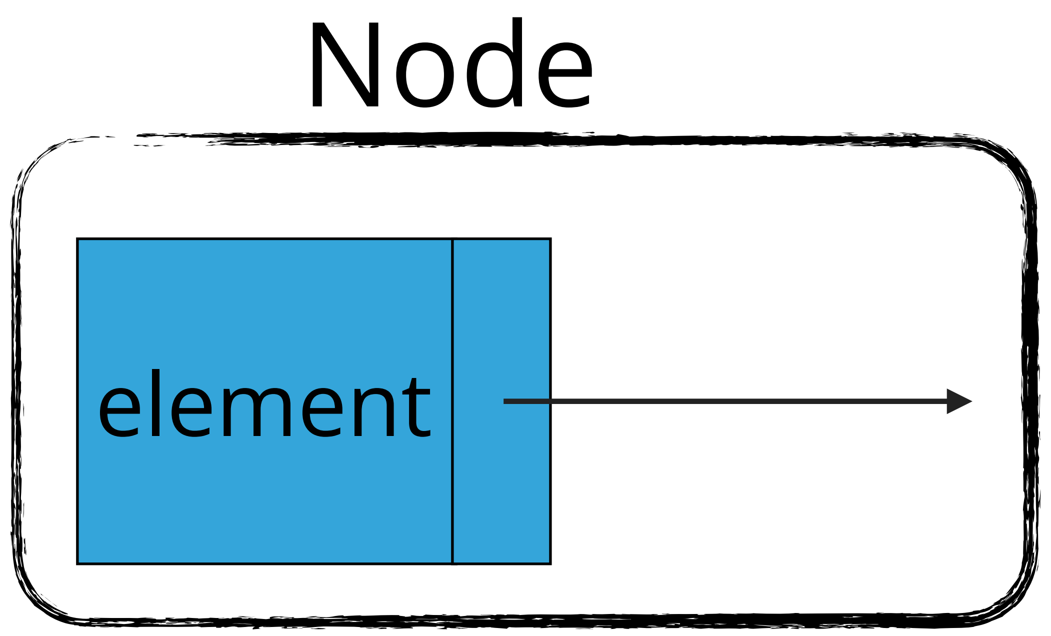


CS	?	!
0	1	2

Recursive Definition of Singly Linked Lists

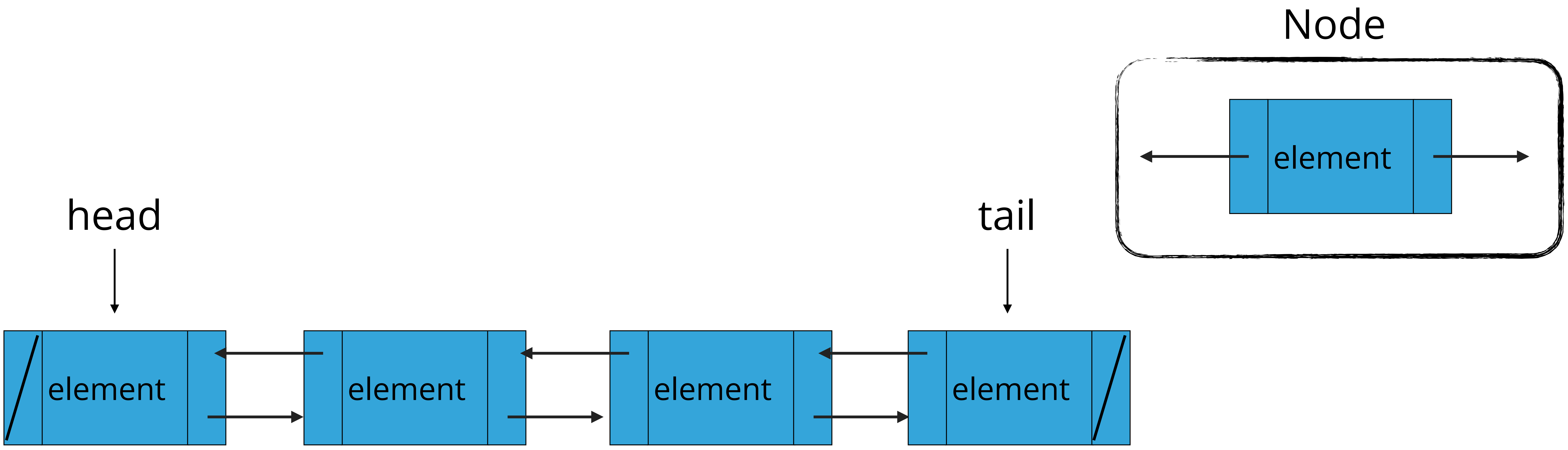
- A singly linked list is either empty (null) or a node having a reference to a singly linked list.
- Node: is a data type that holds elements of the same type and a reference to a next node.





Recursive Definition of Doubly Linked Lists

- A doubly linked list is either empty (null) or a node having a reference to a doubly linked list.
- Node: is a data type that holds any kind of data and two references to the previous and next node.

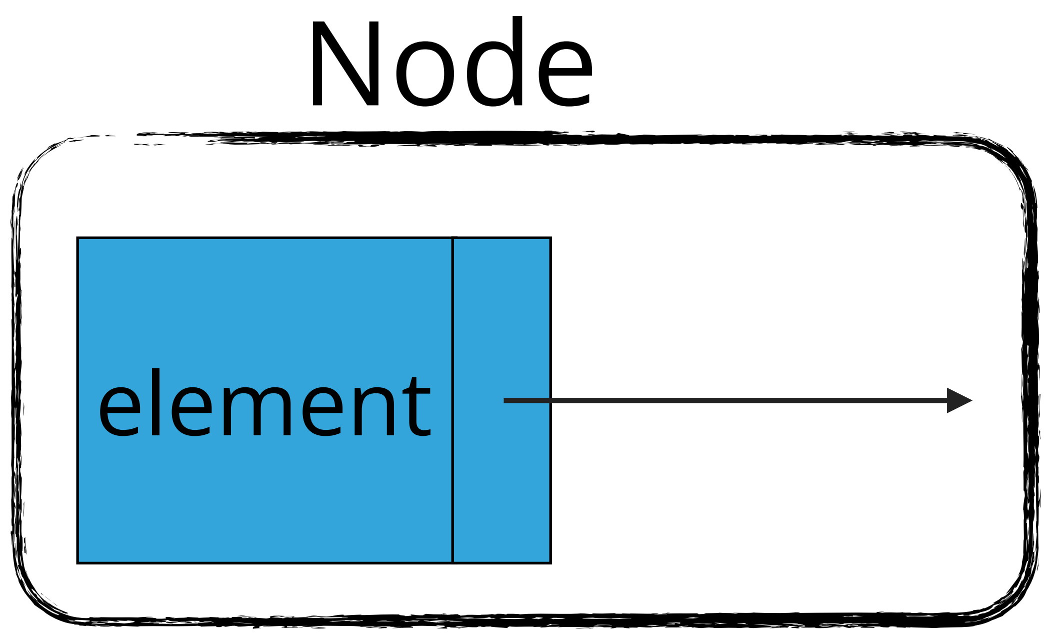


Informal summary of singly & doubly linked lists

- Singly linked lists only have a “next” pointer, and a head
  - add() and remove() happen at the head
- Double linked lists have both a “prev” and “next” pointer, and a head and a tail

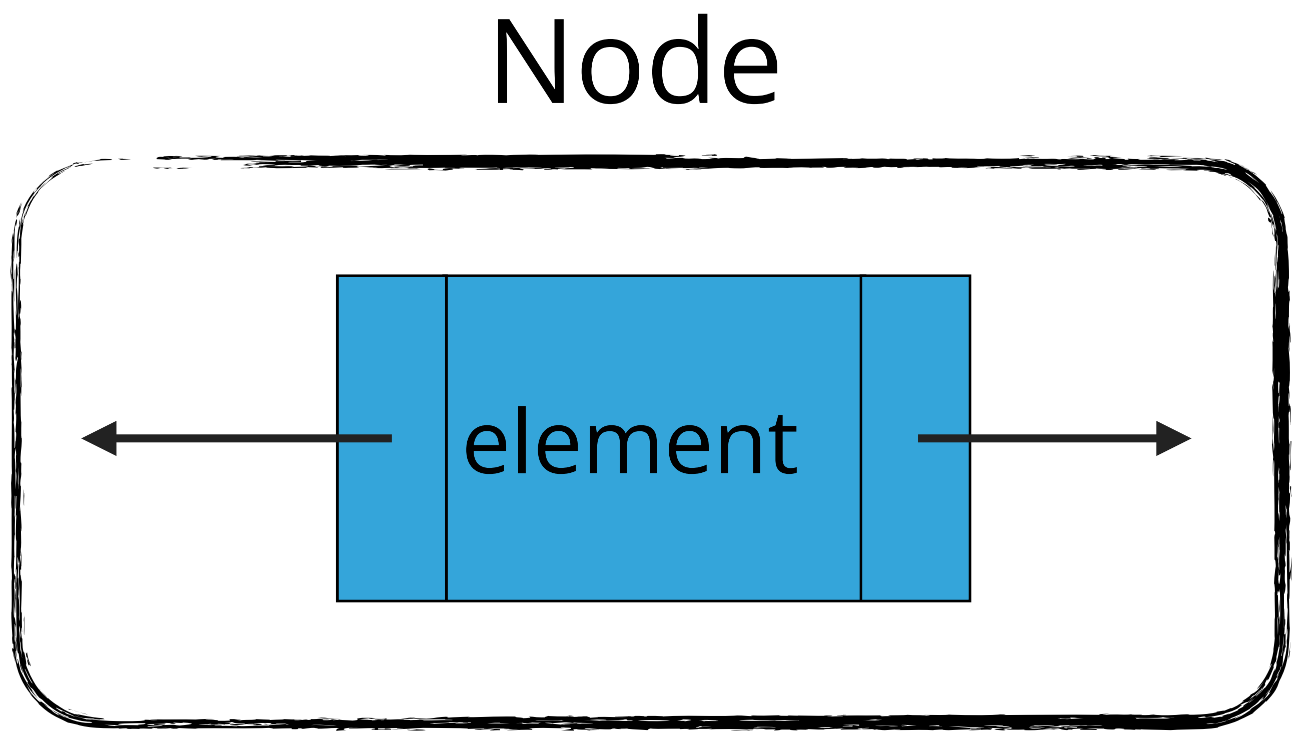
Node (single)

```
private class Node {
    E element;
    Node next;
  }
```



Node (double)

```
private class Node {
   E element;
   Node next;
   Node prev;
 }
```



Singly linked list walkthrough

`SinglyLinkedList()`: Constructs an empty SLL

head = ?

size = ?

What should happen?

```
SinglyLinkedList<String> sll = new SinglyLinkedList<String>();
```

`SinglyLinkedList()`: Constructs an empty SLL

```
SinglyLinkedList<String> sll = new SinglyLinkedList<String>();
```

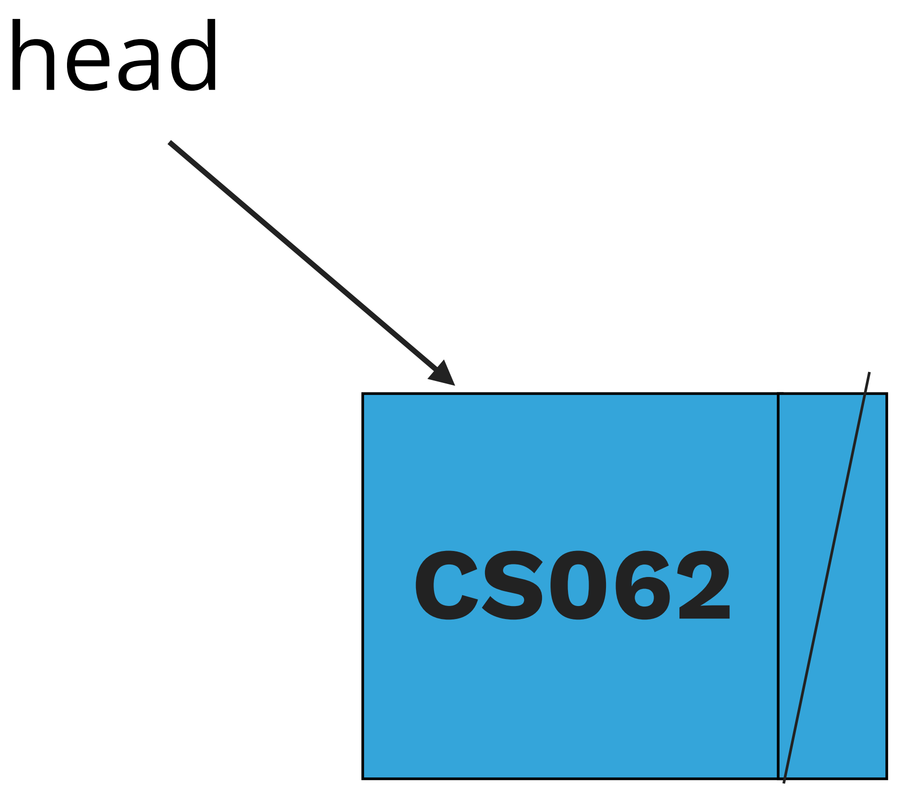
head = null

size = 0

What should happen?

```
sll.add(“CS062”);
```

`add(E element)`: Inserts the specified element at the head of the singly linked list

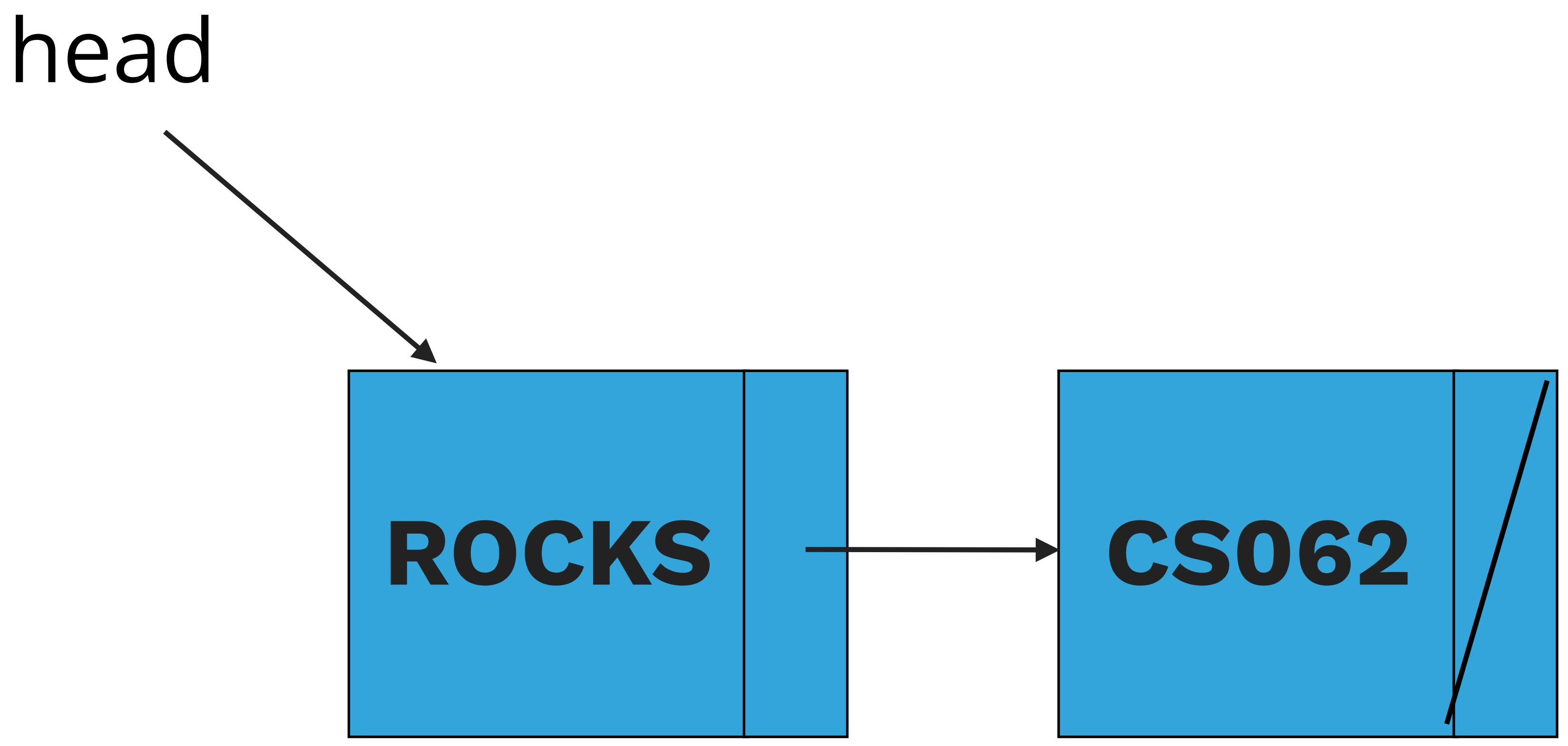


```
sll.add(“CS062”)

size=1
```

What should happen?

```
sll.add(“ROCKS”);
```

`add(E element)`: Inserts the specified element at the head of the singly linked list



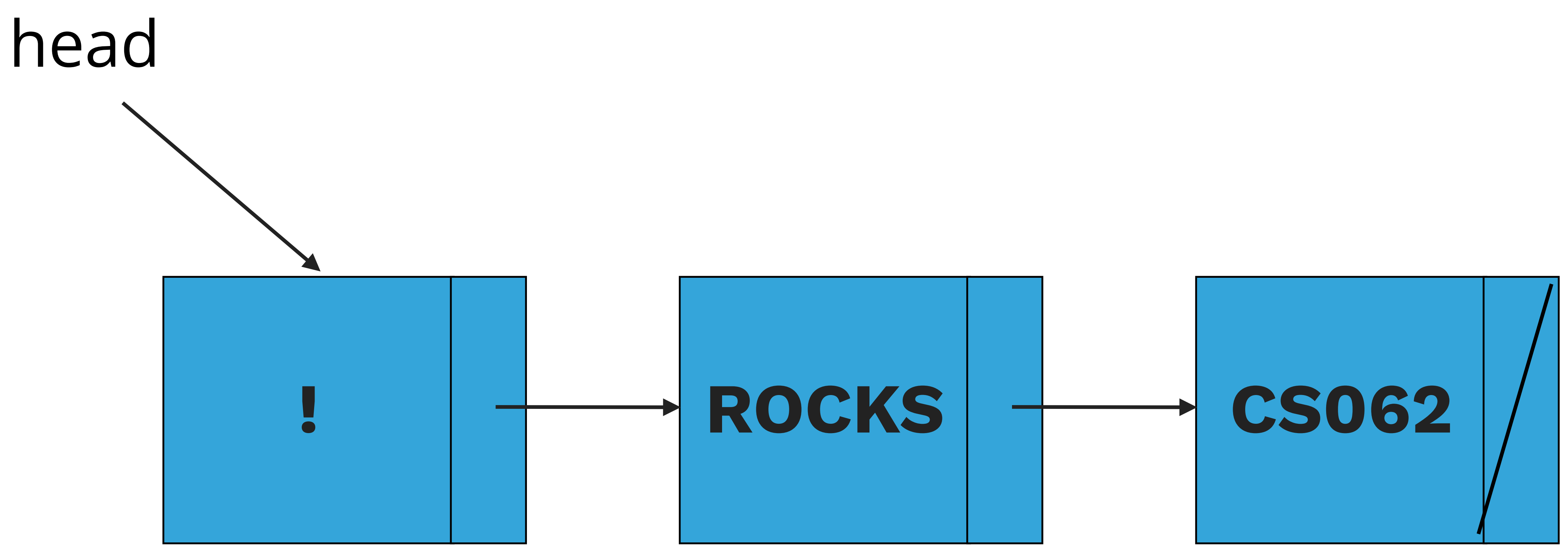
`sll.add("ROCKS")`

`size=2`

What should happen?

`sll.add("!");`

`add(E element)`: Inserts the specified element at the head of the singly linked list



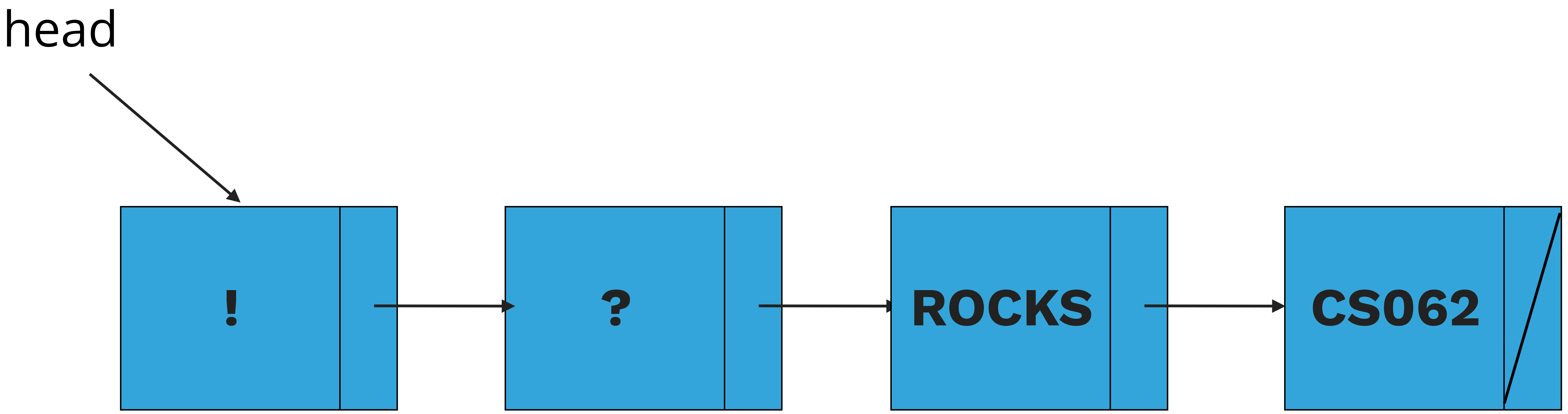
`sll.add(“!”)`

`size=3`

What should happen?

`sll.add(1,“?”);`

`add(int index, E element)`: **Adds element at the specified index**



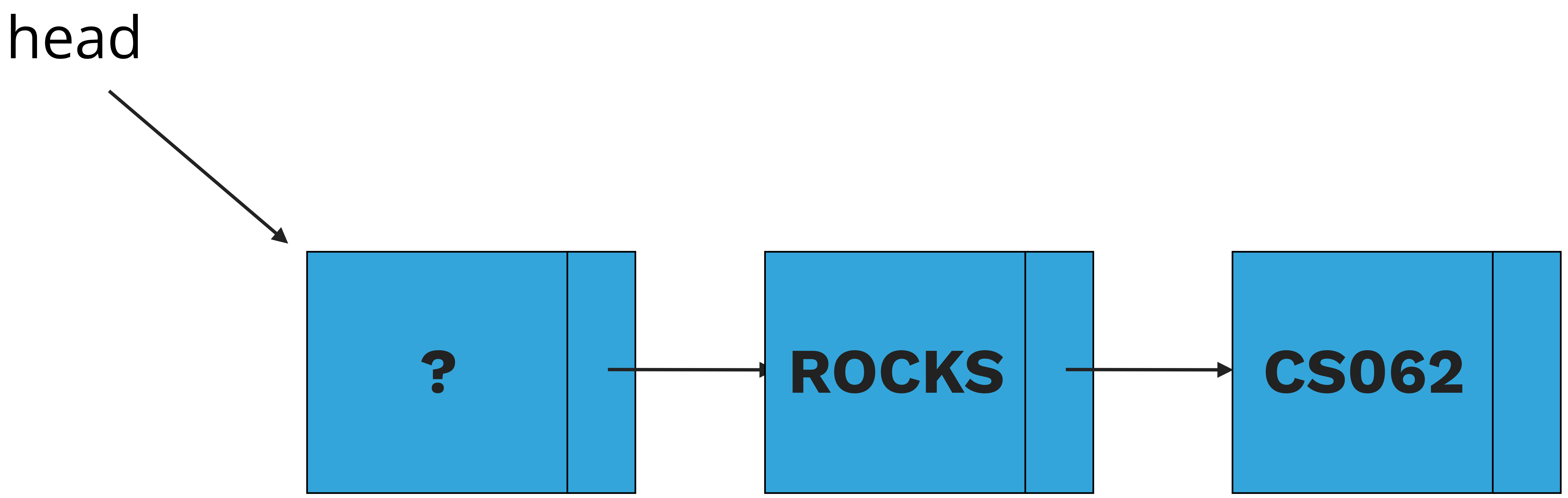
`sll.add(1,“?”)`

`size=4`

What should happen?

`sll.remove();`

`remove()`: Removes and returns the head of the singly linked list

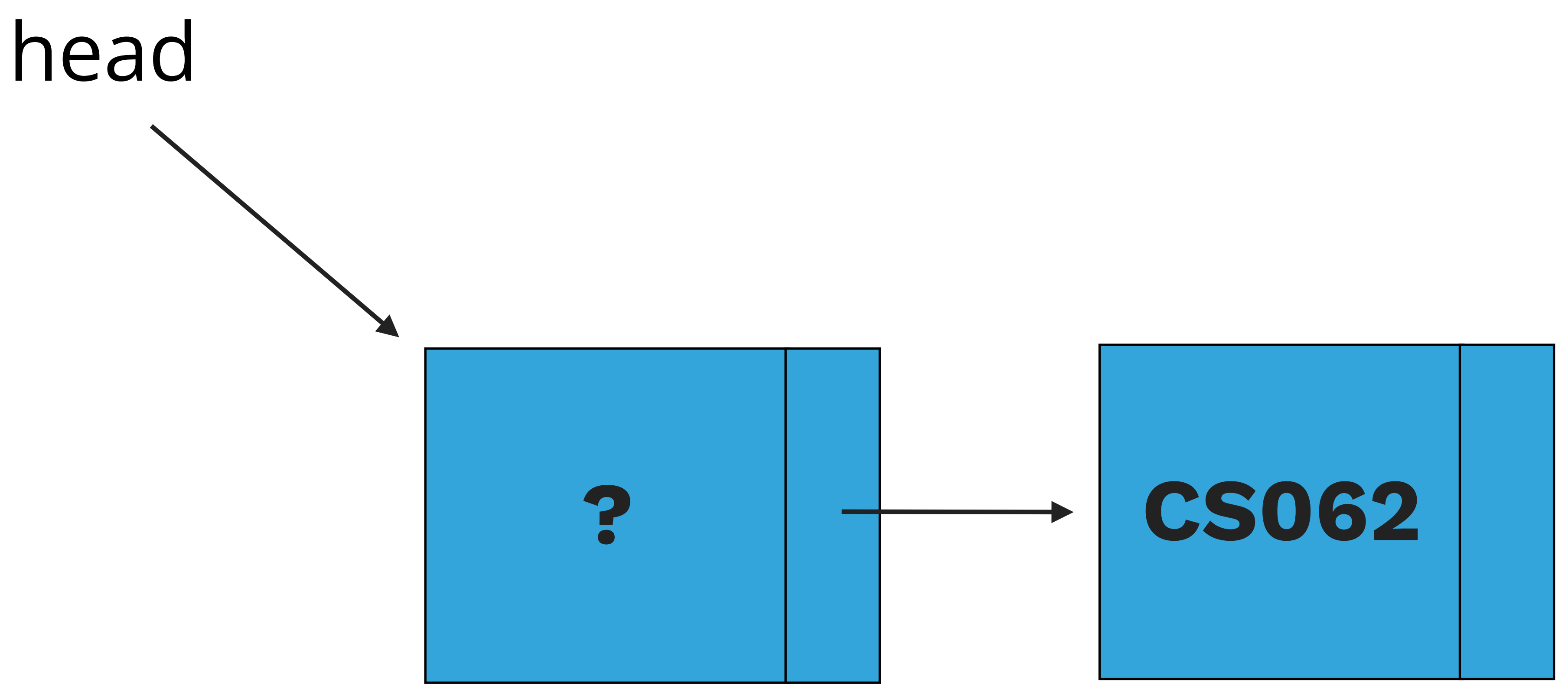


```
sll.remove()

size=3
```

What should happen?

```
sll.remove(1);
```

`remove(int index)`: Removes and returns the element at the specified index



```
sll.remove(1)

size=2
```

Doubly linked list walkthrough

DoublyLinkedList(): Constructs an empty DLL

head

tail

size

What should happen?

```
DoublyLinkedList<String> dll = new DoublyLinkedList<String>();
```

DoublyLinkedList(): Constructs an empty DLL

```
DoublyLinkedList<String> dll = new DoublyLinkedList<String>();
```

head = null

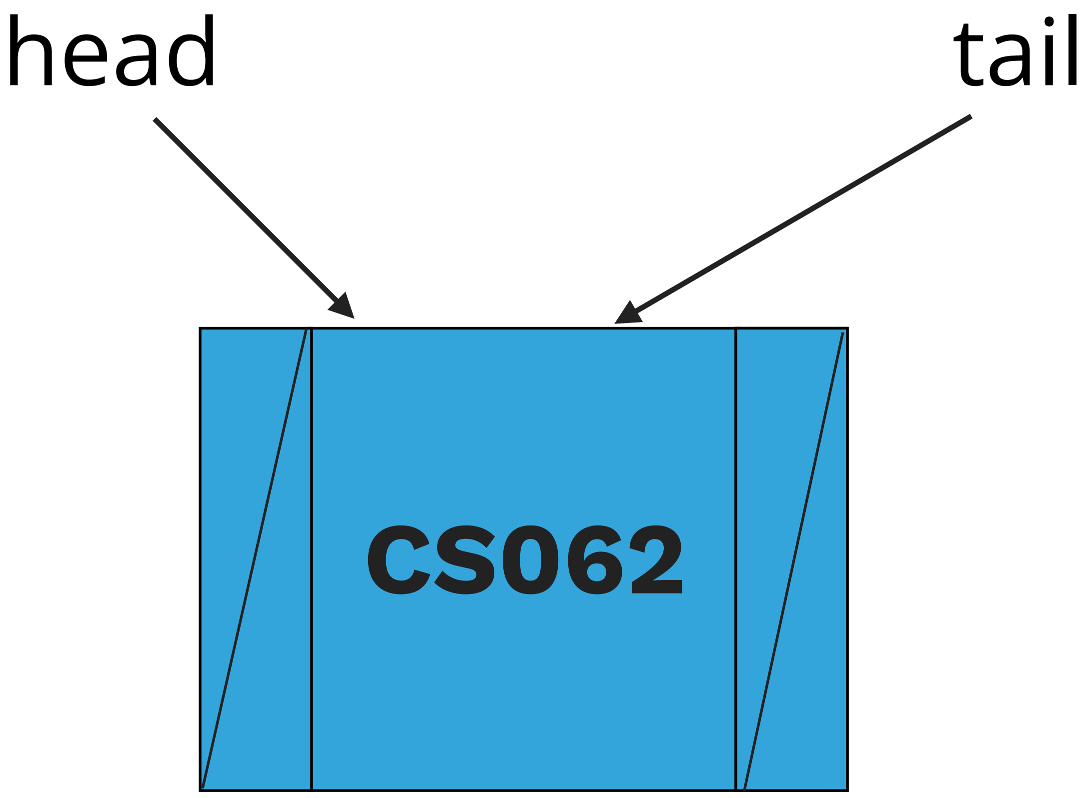
tail = null

size = 0

What should happen?

```
dll.add(“CS062”);
```

`add(E element)`: Inserts the specified element at the head of the doubly linked list.

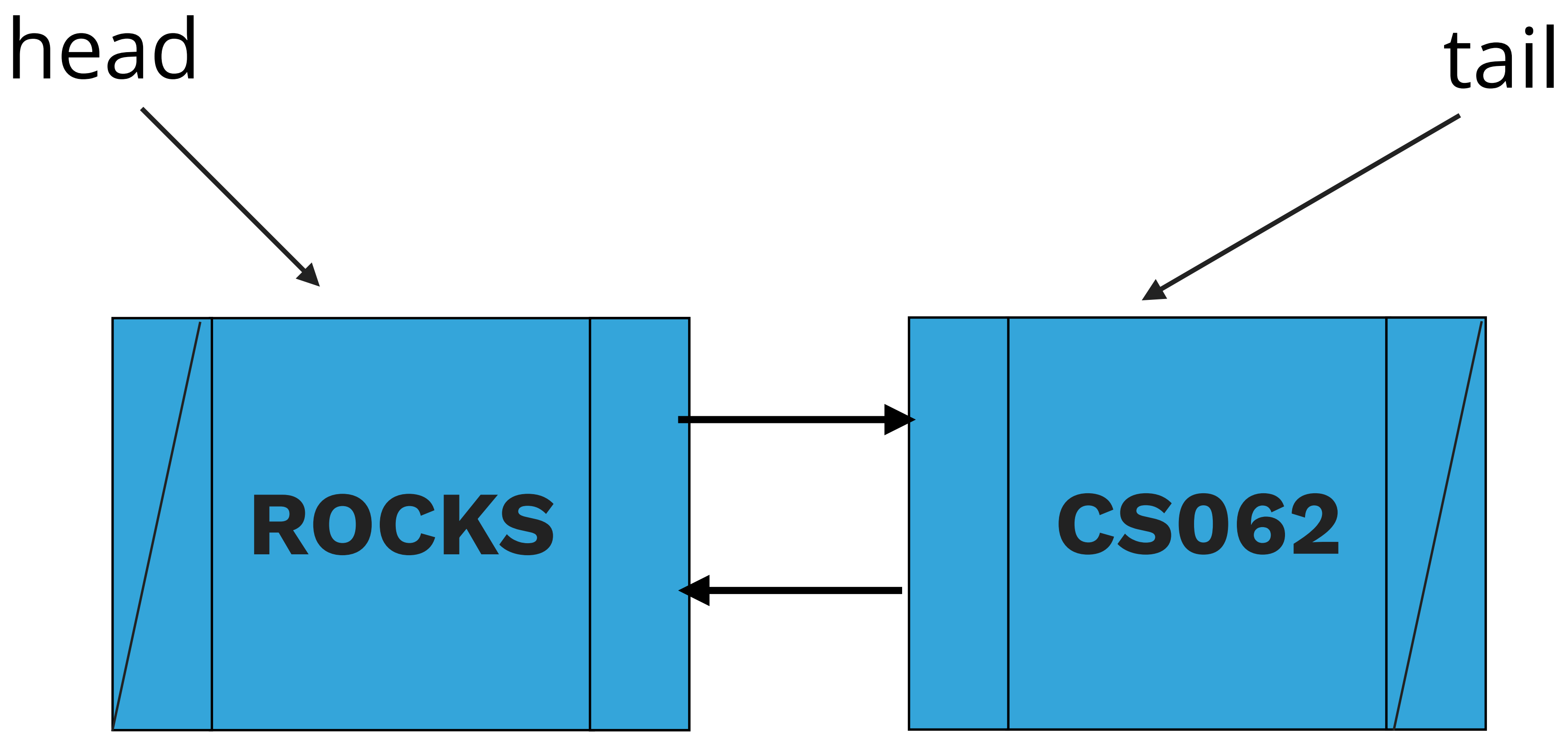


```
dll.add(“CS062”)

size=1
```

What should happen?

```
dll.addFirst(“ROCKS”);
```

`addFirst(E element)`: Inserts the specified element at the head of the doubly linked list



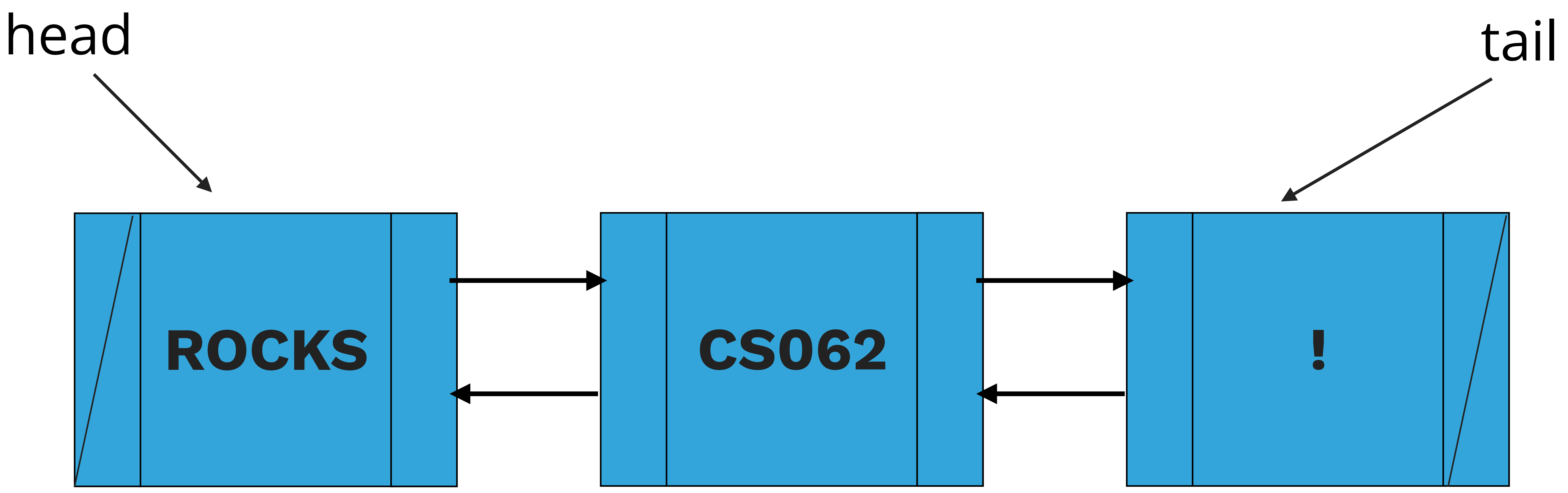
`dll.addFirst(“ROCKS”)`

`size=2`

What should happen?

`dll.addLast(“!”);`

`addLast(E element)`: Inserts the specified element at the tail of the doubly linked list



`dll.addLast("!")`

`size=3`

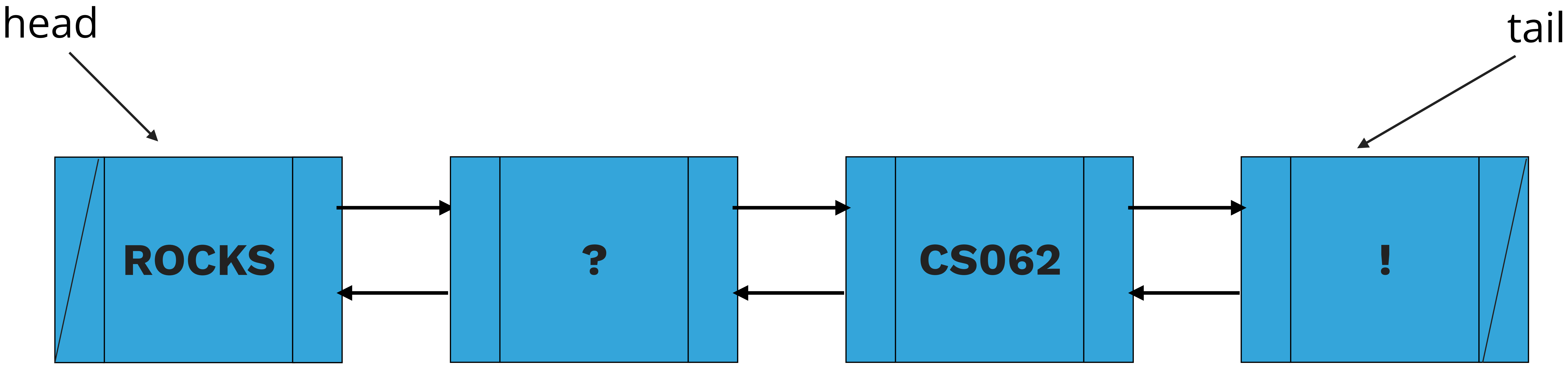
What should happen?

`dll.add(1,"?");`

`add(int index, E element)`: Adds element at the specified index

`dll.add(1,“?”)`

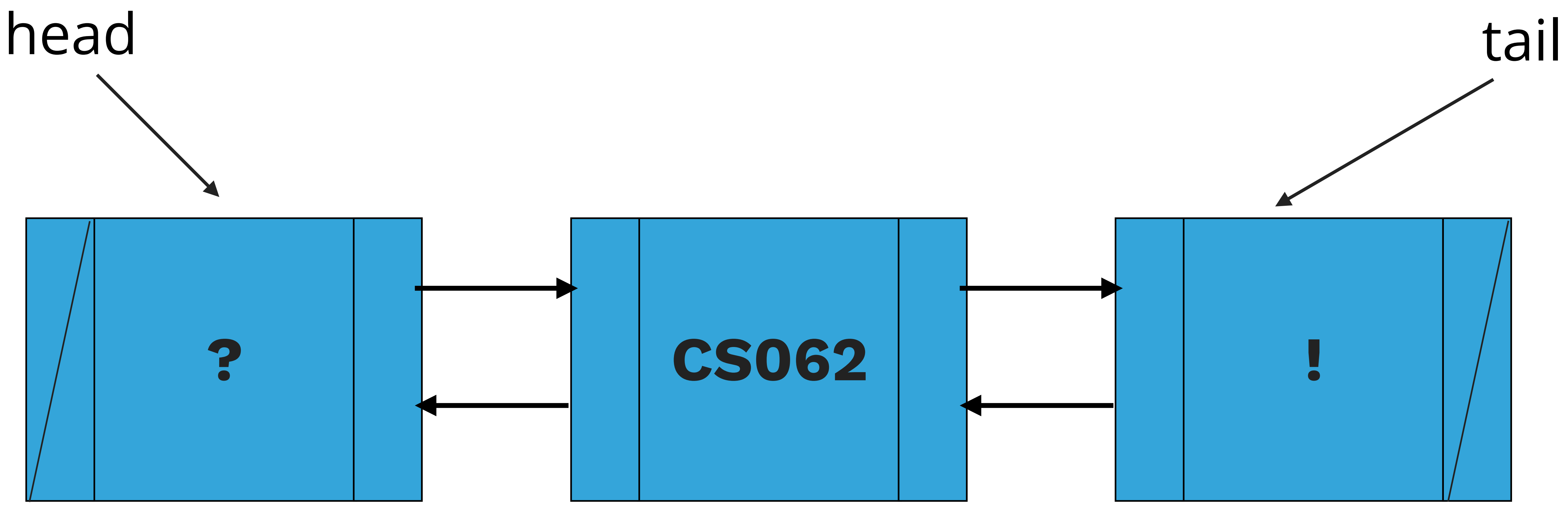
`size=4`



What should happen?

`dll.remove();`

`remove()`: Removes and returns the head of the doubly linked list



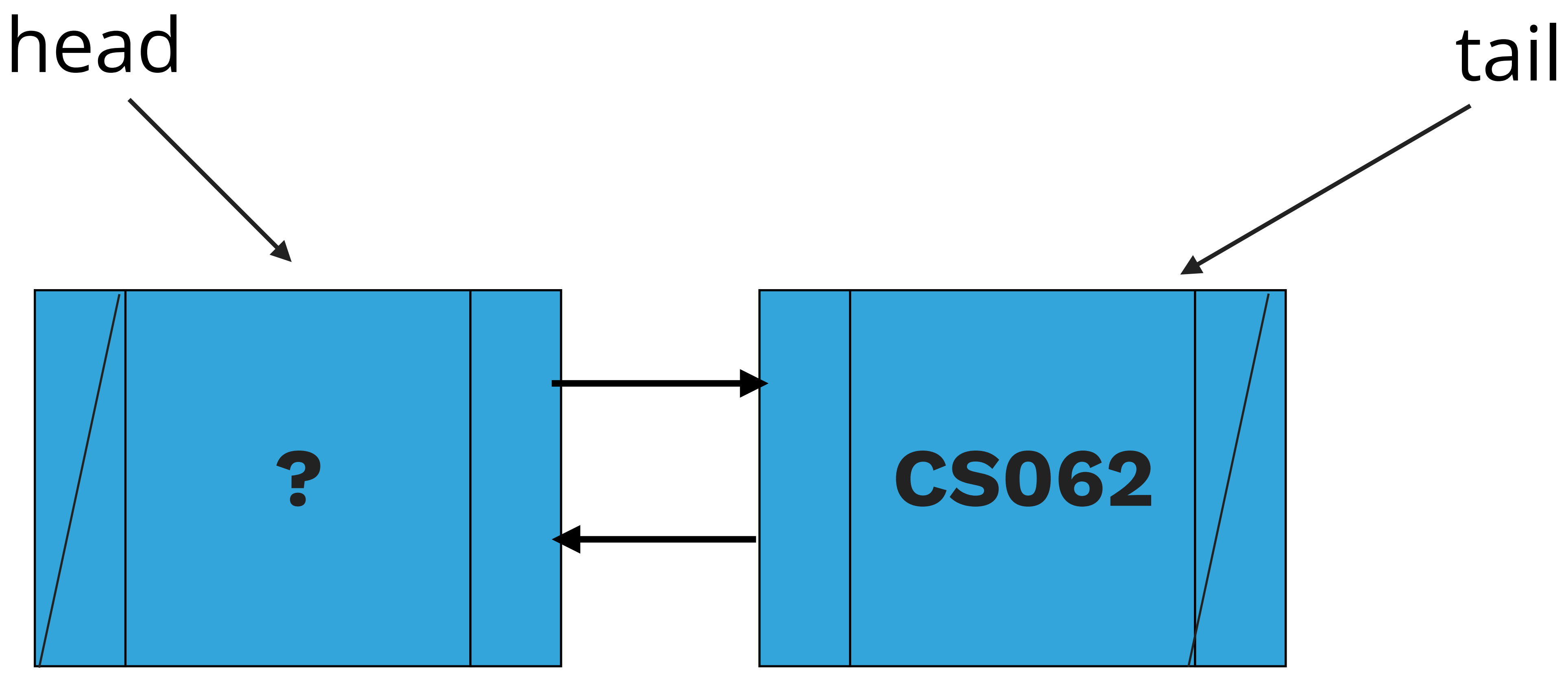
`dll.remove()`

`size=3`

What should happen?

`dll.removeLast();`

`removeLast()`: Removes and returns the tail of the doubly linked list

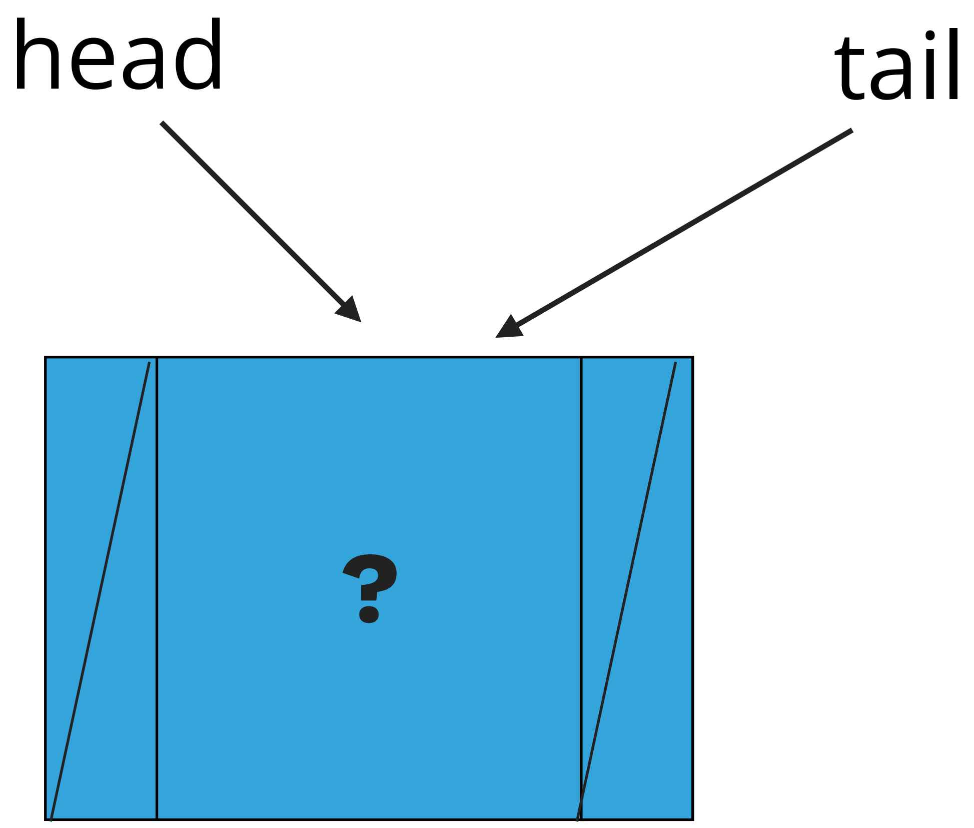


```
dll.removeLast()

size=2
```

What should happen?

```
dll.remove(1);
```

`remove(int index)`: Removes and returns the element at the specified index



```
dll.remove(1)

size=1
```

Worksheet time!

- Do Qs 1-4 on your worksheet.

Suppose x is a reference to a Node and that node is not the last one on the linked list. What is the effect of the following code fragment?

```
x.next = x.next.next;
```

Suppose x and t are references to different Nodes. What is the effect of the following code fragment?

```
t.next = x.next;

x.next = t;
```

Suppose x and t are references to different Nodes in a doubly linked list. What is the effect of the following code fragment?

```
t.prev = x;
t.next = x.next;
x.next.prev = t;
x.next = t;
```

What if the code was in a different order?

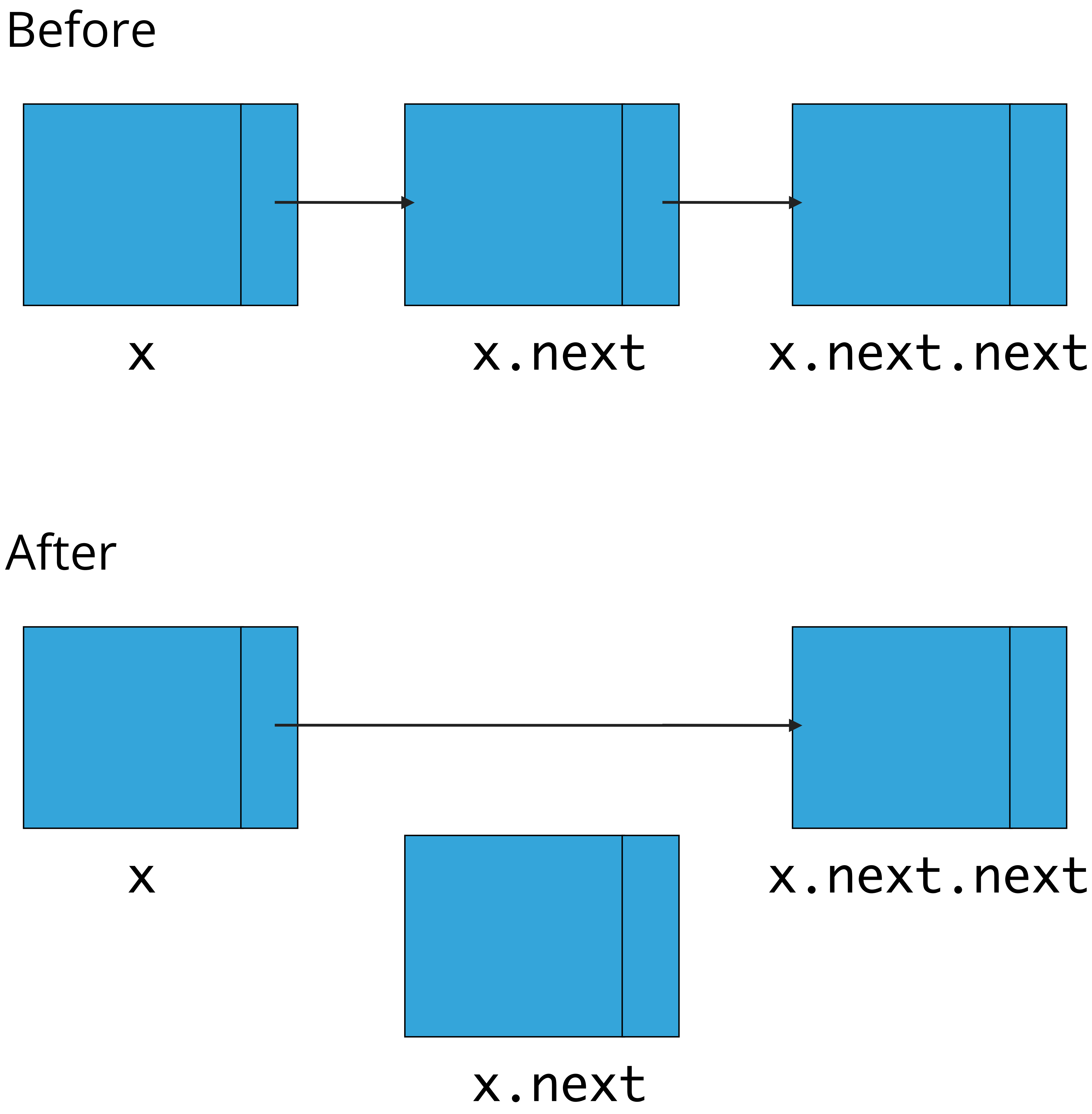
```
x.next = t;
x.next.prev = t;
t.next = x.next;
t.prev = x;
```

Worksheet answers

Suppose x is a reference to a Node and that node is not the last one on the linked list. What is the effect of the following code fragment?

```
x.next = x.next.next;
```

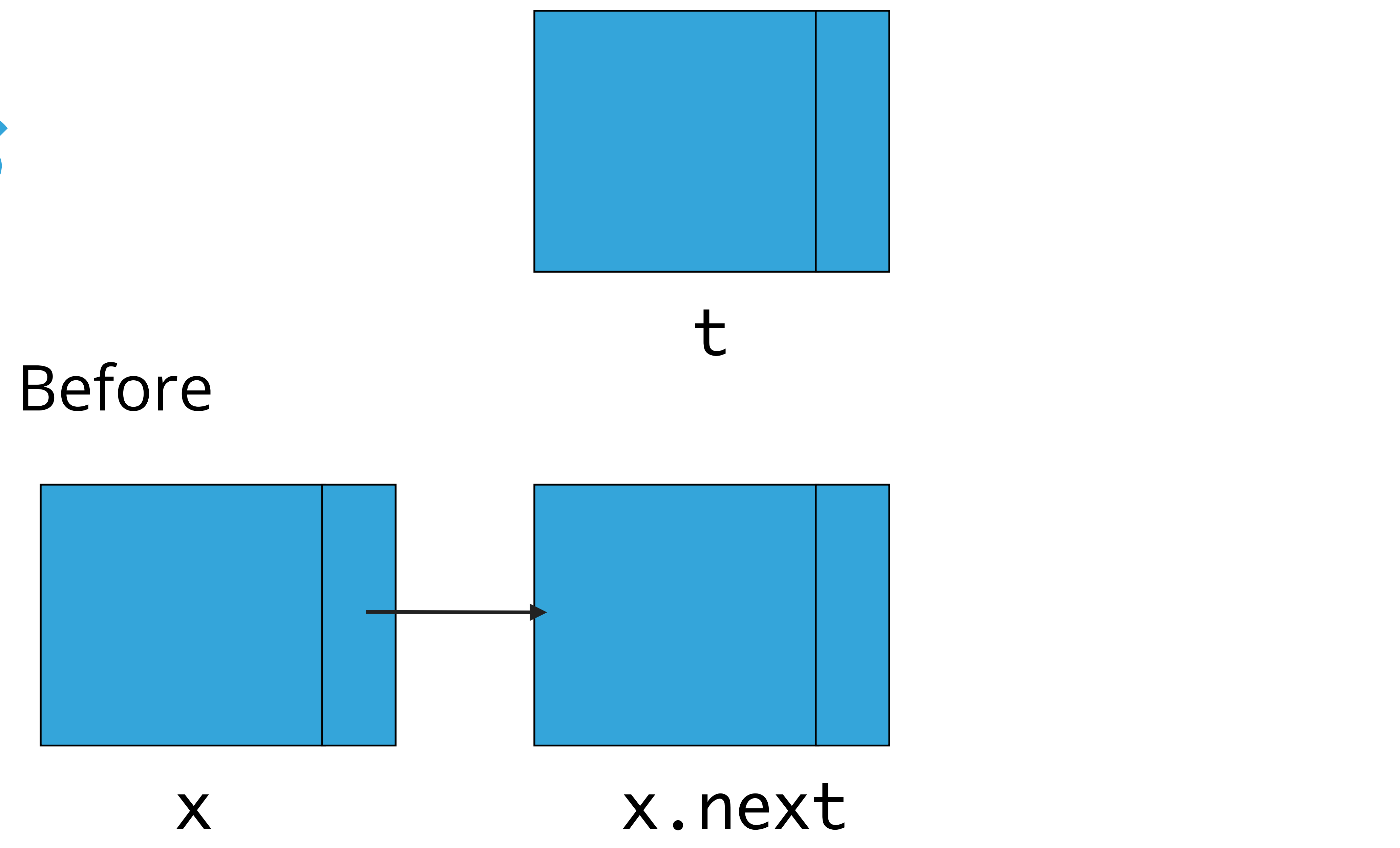
It removes the node after x.



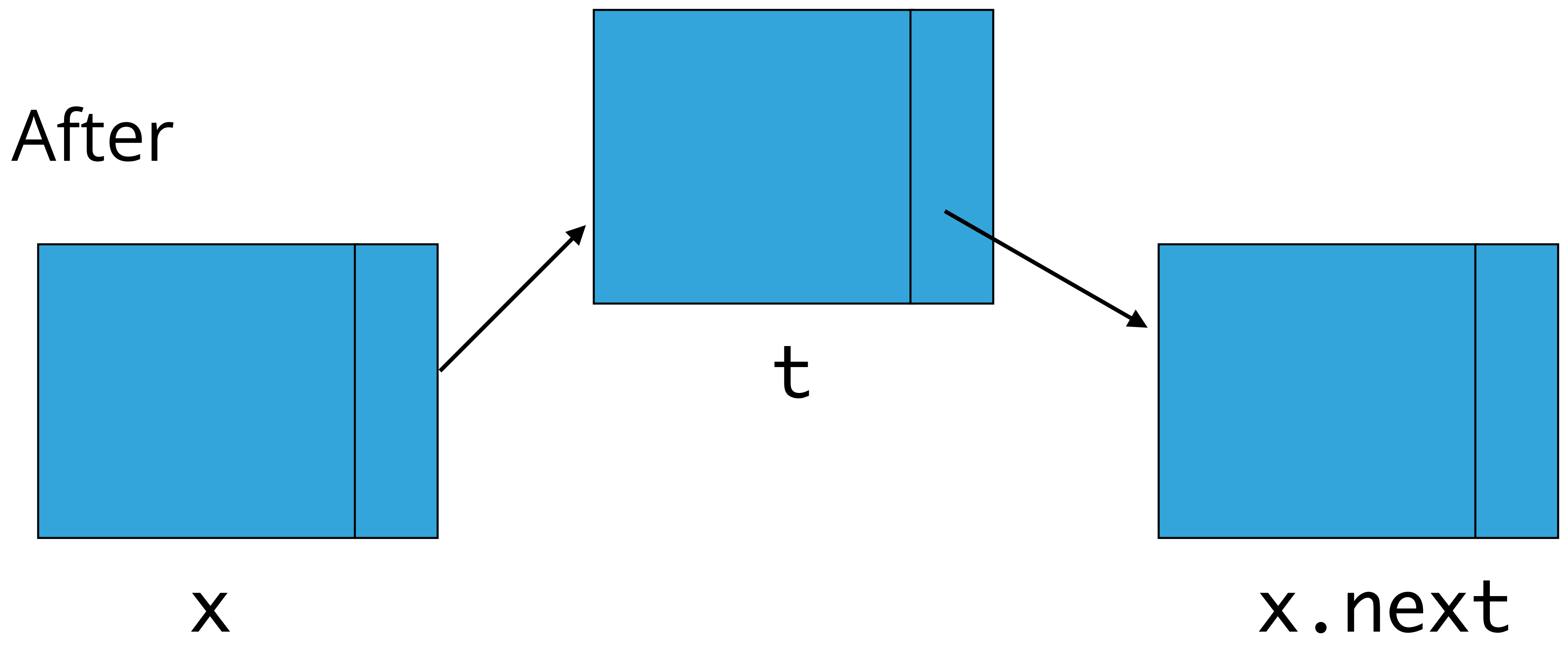
Worksheet answers

Suppose x and t are references to different Nodes. What is the effect of the following code fragment?

```
t.next = x.next;

x.next = t;
```



It inserts t between x and x.next.

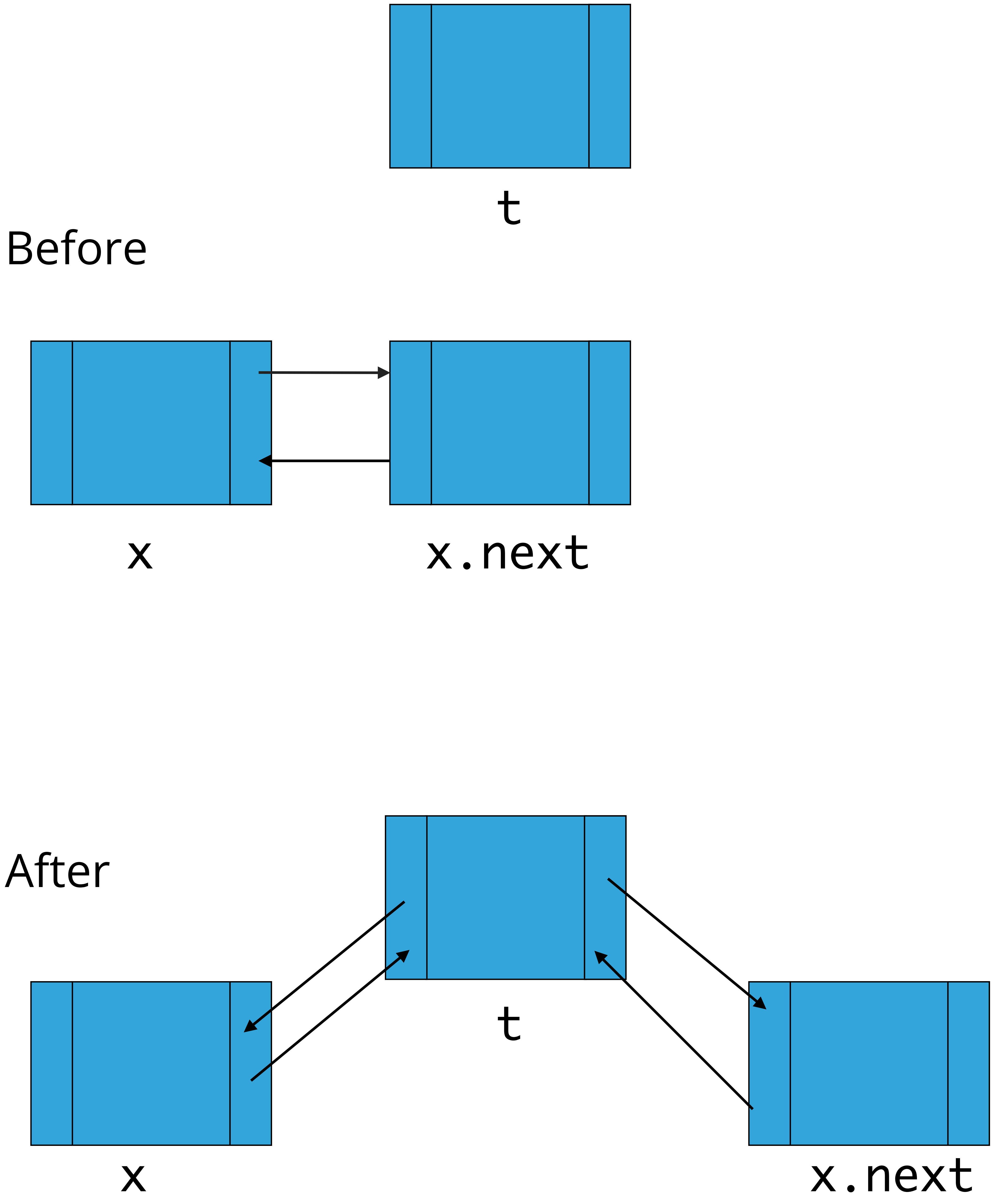


Worksheet answers

Suppose x and t are references to different Nodes in a doubly linked list. What is the effect of the following code fragment?

```
t.prev = x;
t.next = x.next;
x.next.prev = t;
x.next = t;
```

It inserts t between x and x.next.

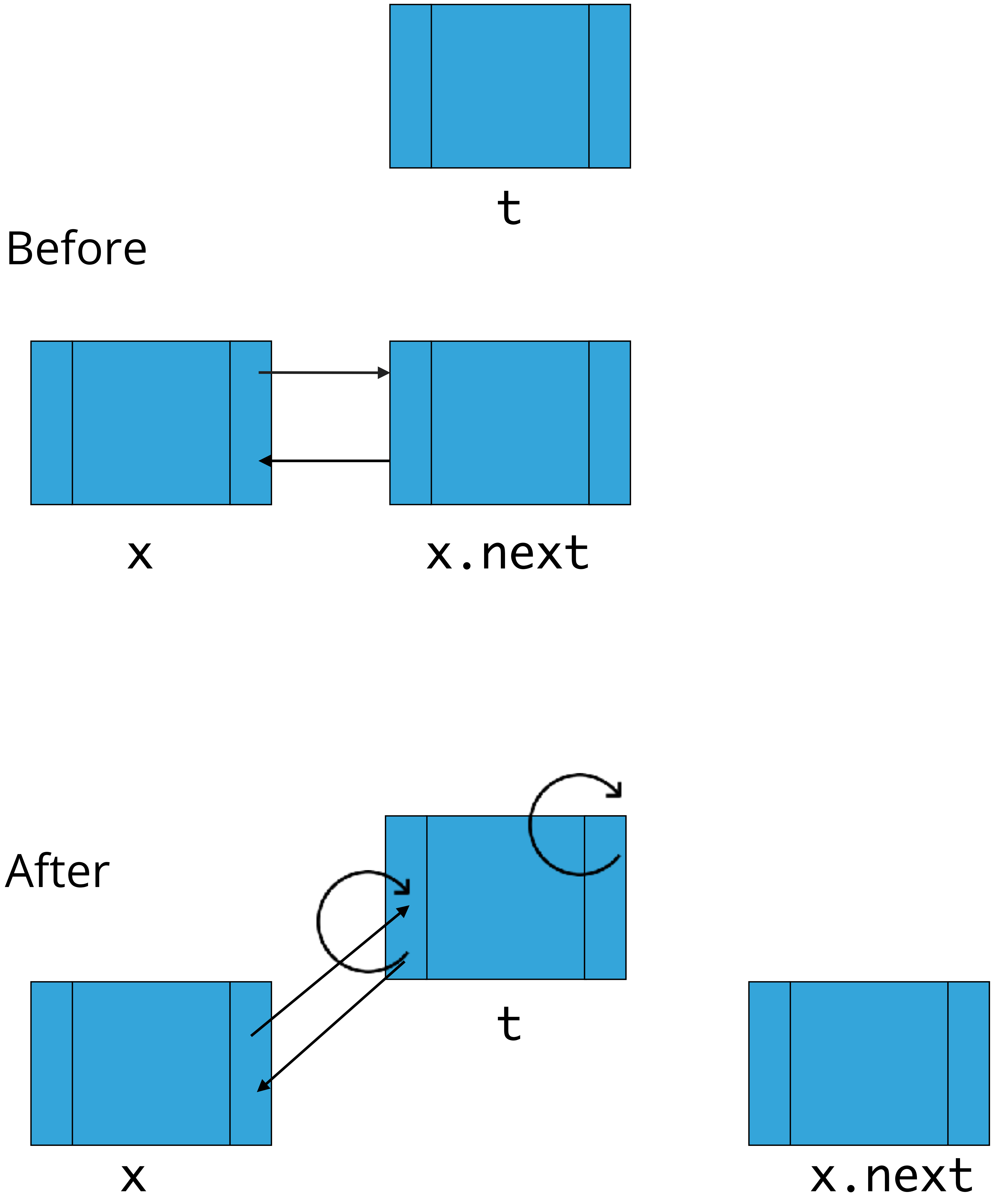


Worksheet answers

What if the code was in a different order?

```
x.next = t;
x.next.prev = t;
t.next = x.next;
t.prev = x;
```

It’s gibberish



Implementation

Reminder: Interface List

```
public interface List <E> {
    void add(E element);
    void add(int index, E element);
    void clear();
    E get(int index);
    boolean isEmpty();
    E remove();
    E remove(int index);
    E set(int index, E element);
    int size();
}
```

Linked Lists: Standard Operations

- `SinglyLinkedList()`: Constructs an empty singly linked list.
- `DoublyLinkedList()`: Constructs an empty doubly linked list.
- `isEmpty()`: Returns true if the linked list does not contain any element.
- `size()`: Returns the number of elements in the linked list.
- `E get(int index)`: Returns the element at the specified index.
- `add(E element)`: Inserts the specified element at the head of the linked list.
  - `addFirst(E element)`: Inserts the specified element at the head of the **doubly** linked list.
  - `addLast(E element)`: Inserts the specified element at the tail of the **doubly** linked list.
- `add(int index, E element)`: Inserts the specified element at the specified index.
- E `set(int index, E element)`: Replaces the specified element at the specified index and returns the old element
- `E remove()`: Removes and returns the head of the linked list.
  - `E removeLast()`: Removes and returns the tail of the **doubly** linked list.
- `E remove(int index)`: Removes and returns the element at the specified index.
- `clear()`: Removes all elements.

Our own implementation of Linked Lists

- We will follow the recommended textbook style.
  - It does not offer a class for this so we will build our own.
- We will work with generics because we want singly linked lists to hold elements of any type (as long as they are of the same type).
- We will implement the List interface we defined in past lectures.
- We will use an inner class Node and we will keep track of how many elements we have in our linked list (both single & double).

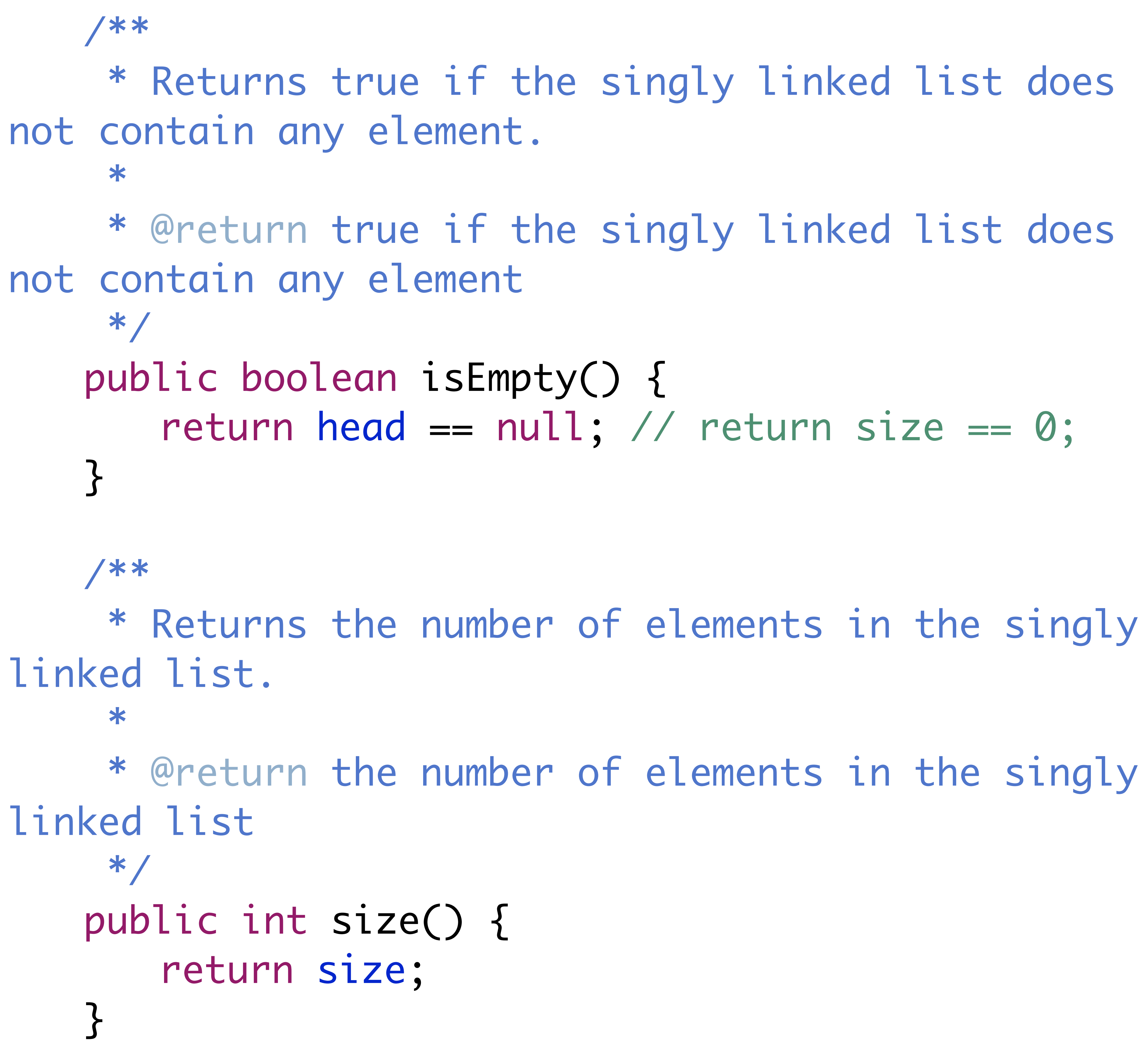
SLL: Instance variables and inner class

```
public class SinglyLinkedList<E> implements List<E>{
   private Node head; // head of the singly linked list
   private int size; // number of nodes in the singly linked list

   /**
    * This nested class defines the nodes in the singly linked list with a value
    * and pointer to the next node they are connected.
    */
   private class Node {
      E element;
      Node next;
   }
```

DLL: Instance variables and inner class

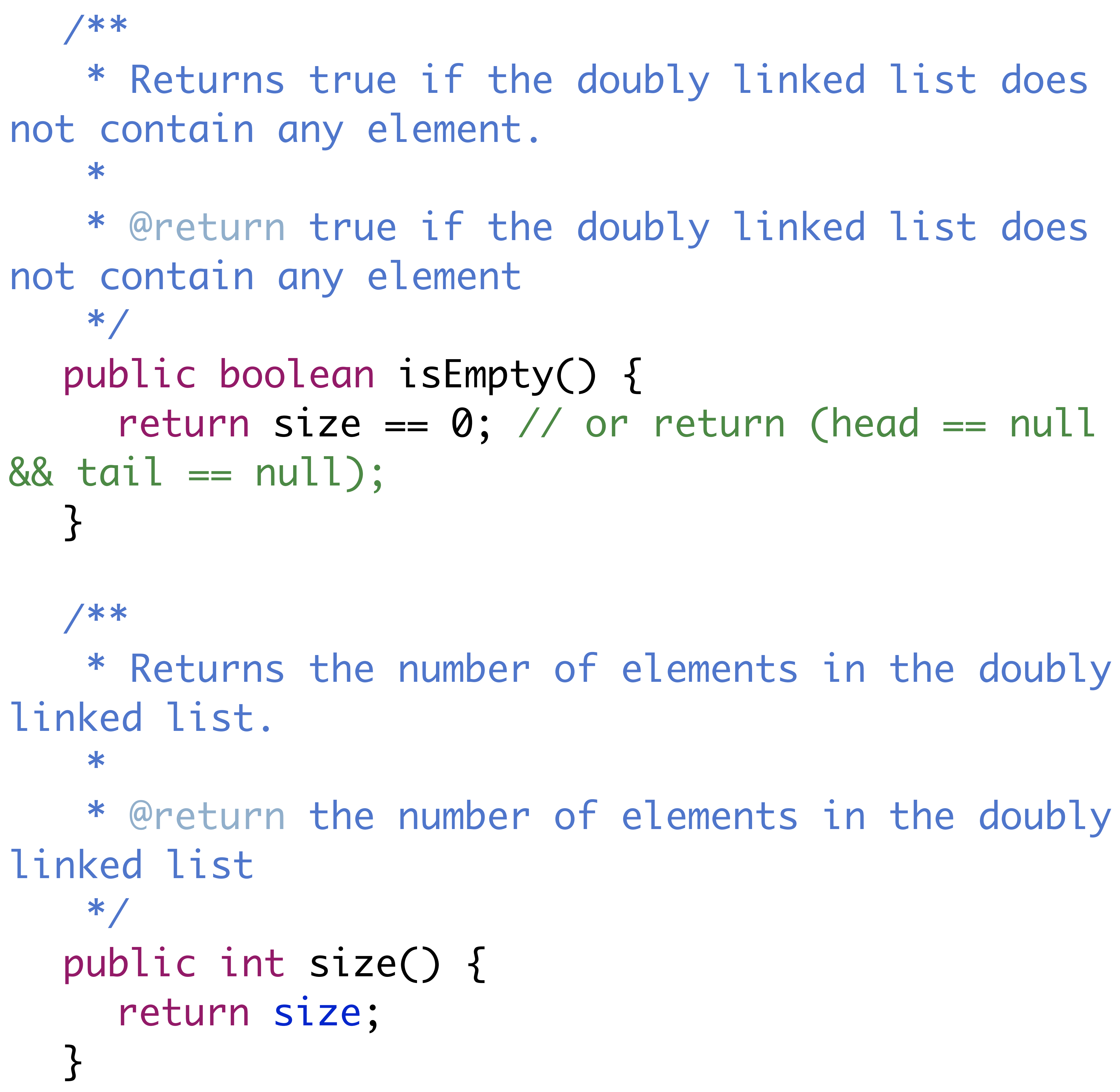
```
public class DoublyLinkedList<E> implements List<E> {
  private Node head; // head of the doubly linked list
  private Node tail; // tail of the doubly linked list
  private int size; // number of nodes in the doubly linked list

  /**
   * This nested class defines the nodes in the doubly linked list with a value
   * and pointers to the previous and next node they are connected.
   */
  private class Node {
    E element;
    Node next;
    Node prev;
  }
```

Check if is empty and how many elements



```java
    /**
     * Returns true if the singly linked list does
not contain any element.
     *
     * @return true if the singly linked list does
not contain any element
     */
    public boolean isEmpty() {
        return head == null; // return size == 0;
    }

    /**
     * Returns the number of elements in the singly
linked list.
     *
     * @return the number of elements in the singly
linked list
     */
    public int size() {
        return size;
    }
```

SLL



```java
    /**
     * Returns true if the doubly linked list does
not contain any element.
     *
     * @return true if the doubly linked list does
not contain any element
     */
    public boolean isEmpty() {
        return size == 0; // or return (head == null
&& tail == null);
    }

    /**
     * Returns the number of elements in the doubly
linked list.
     *
     * @return the number of elements in the doubly
linked list
     */
    public int size() {
        return size;
    }
```

DLL

SLL: Retrieve element from specified index

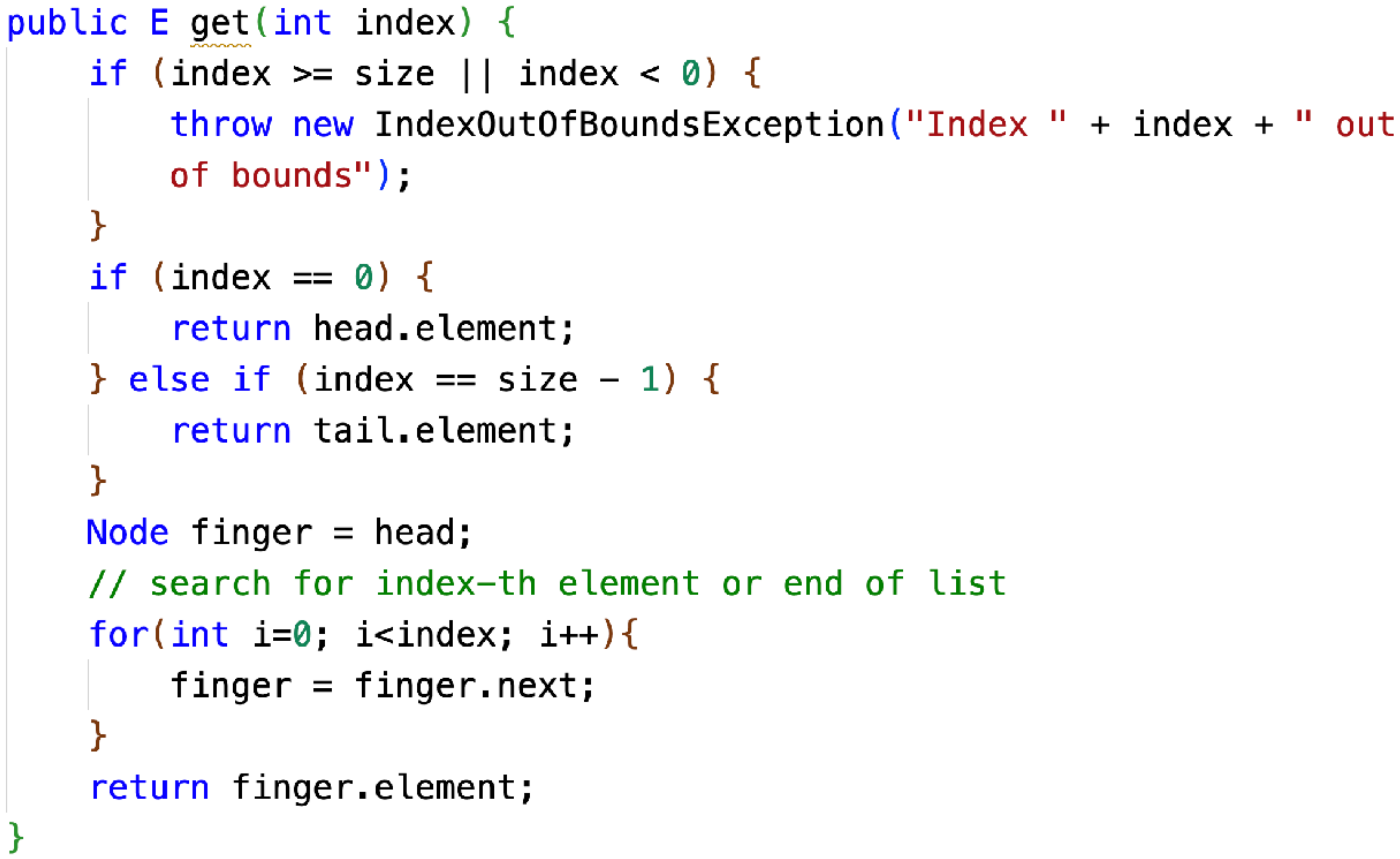
```
  /**
 * Returns element at the specified index.
 *
 * @param index
 *            the index of the element to be returned
 * @return the element at specified index
 * @pre 0<=index<size
 */
public E get(int index) {
   // check whether index is valid
      if (index >= size || index < 0){
         throw new IndexOutOfBoundsException("Index " + index + " out of bounds");
     }
   // set a temporary pointer to the head
   Node finger = head;
   // search for index-th element or end of list
   for (int i=0; i< index; i++){
      finger = finger.next;
   }
   // return the element stored in the node that the temporary pointer points to
   return finger.element;
}
```

Worksheet time!

- Write public E get(int index) for doubly linked lists.

```
/**
 * Returns element at the specified index.
 *
 * @param index
 *            the index of the element to be returned
 * @return the element at specified index
 */
public E get(int index) {
  // check whether index is valid

     // if index is 0, return element at head


      // else if index is size-1, return element at tail


  // set a temporary pointer to the head

  // search for index-th element or end of list


  // return the element stored in the node that the temporary pointer points to
}
```

Worksheet answers



```
public E get(int index) {
    if (index >= size || index < 0) {
        throw new IndexOutOfBoundsException("Index " + index + " out
        of bounds");
    }
    if (index == 0) {
        return head.element;
    } else if (index == size - 1) {
        return tail.element;
    }
    Node finger = head;
    // search for index-th element or end of list
    for(int i=0; i<index; i++){
        finger = finger.next;
    }
    return finger.element;
}
```

Insert element at head of singly linked list

```
  /**
 * Inserts the specified element at the head of the singly linked list.
 *
 * @param element
 *            the element to be inserted
 */
public void add(E element) {
   // Create a pointer to head
   Node oldHead = head;

   // Make a new node that will hold the element and assign it to head.
   head = new Node();
   head.element = element;
   // fix pointers
   head.next = oldHead;
   // increase number of nodes
   size++;
}
```

Insert element at head of doubly linked list

```
/**
 * Inserts the specified element at the head of the doubly linked list.
 *
 * @param element the element to be inserted
 */
public void addFirst(E element) {
  // Create a pointer to head
  Node oldHead = head;

  // Make a new node and assign it to head. Fix pointers and update element
  head = new Node();
  head.element = element;
  head.next = oldHead;
  head.prev = null;

  // if first node to be added, adjust tail to it.
  if (tail == null){
    tail = head;
      }
  else{
     // else fix previous pointer to head
    oldHead.prev = head;
      }
  // increase number of nodes in doubly linked list.
      size++;
 }
```

new: set prev of new head to null

new: is the DLL just one node?

new: set prev of old head to our new head

Insert element at a specified index (SLL)

```
public void add(int index, E element) {
         // check that index is within range
      if (index > size || index < 0){
             throw new IndexOutOfBoundsException("Index " + index + " out of bounds");
         }

         // if index is 0, then call one-argument add
   if (index == 0) {
       add(element);
         // else
   } else {
       // make two pointers, previous and finger. Point previous to null and finger to head
       Node previous = null;
       Node finger = head;
       // search for index-th position by pointing previous to finger and advancing finger
       for (int i=0; i<index; i++){
          previous = finger;
          finger = finger.next;
       }
       // create new node to insert in correct position. Set its pointers and contents
       Node current = new Node();
       current.next = finger;
       current.element = element;
       // make previous point to newly created node.
       previous.next = current;
       // increase number of nodes
       size++;
   }
}
```

When we are at the correct index to insert, finger points to the new element's next element, while previous pointers to the new element's previous element (we are inserting between previous and finger)

Lecture 8 wrap-up

- Exit ticket: https://forms.gle/SjKvLVvvx3zMxNQH8
- Part II of Darwin (everything else) due next Tues 11:59pm
- We'll start next lecture with writing remove at an index for DLLs

Resources

- Linked lists from the textbook: https://algs4.cs.princeton.edu/13stacks/
- See slides following this for more practice problems
- Exercise to the reader: what are the run times for doubly linked list methods? Do they change from singly linked lists?

Bonus practice problem

- Add a deleteLast method in the SinglyLinkedList class that removes the last node of a singly linked list. Think of edge cases.

Bonus answer

```
public void deleteLastNode() {
       if (!isEmpty()) {
           if (size == 1) {
               head = null;
           } else {
               Node current = head;
               for (int i = 0; i < size - 2; i++) {
                   current = current.next;
               }
               current.next = null;
           }
           size--;
       }
   else{//throw some appropriate exception}
   }
```