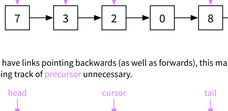
Motivation for Doubly-Linked Lists

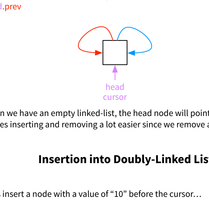
With a single-link linked list, we need to keep track of a precursor and tail to make common operations faster.



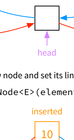
If we have links pointing backwards (as well as forwards), this makes keeping track of precursor unnecessary.

If we want the precursor, we can just access cursor.prev

Doubly-Linked List Representation



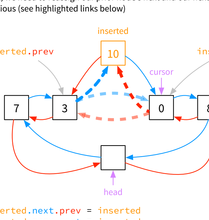
By making an empty (or "dummy") head node, we can access the first element with head.next. Similarly, the last element can be accessed with head.prev



When we have an empty linked-list, the head node will point to itself. This makes inserting and removing a lot easier since we remove all edge cases.

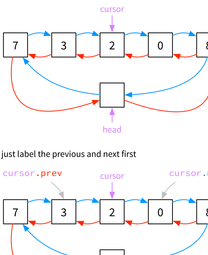
Insertion into Doubly-Linked List

Let's insert a node with a value of "10" before the cursor…

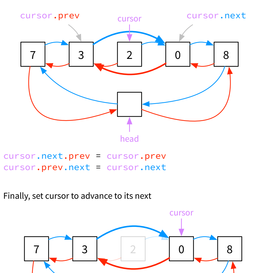


First, let's create a new node and set its links

```
inserted = new Node<E>(element, cursor.prev, cursor)
```



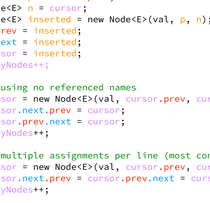
Next, we need to reassign our prior node's next and our next node's previous (see highlighted links below)



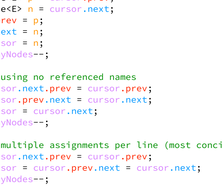

```
inserted.next.prev = inserted
inserted.prev.next = inserted
```

Removal from Doubly-Linked List

We now want to write code which removes the cursor and advances it to the next node. We assume that cursor is valid



Let's just label the previous and next first



Now let's set their links. NOTE: do not change cursor here


```
cursor.next.prev = cursor.prev
cursor.prev.next = cursor.next
```

Finally, set cursor to advance to its next


```
cursor = cursor.next
```

Insert:

```
// using referenced names
Node<E> p = cursor.prev;
Node<E> n = cursor;
Node<E> inserted = new Node<E>(val, p, n);
n.prev = inserted;
p.next = inserted;
cursor = inserted;
manyNodes++;

// using no referenced names
cursor = new Node<E>(val, cursor.prev, cursor);
cursor.next.prev = cursor;
cursor.prev.next = cursor;
manyNodes++;

// multiple assignments per line (most concise)
cursor = new Node<E>(val, cursor.prev, cursor);
cursor.next.prev = cursor.prev.next = cursor;
manyNodes++;
```

Remove:

```
// using referenced names
Node<E> p = cursor.prev;
Node<E> n = cursor.next;
n.prev = p;
p.next = n;
cursor = n;
manyNodes--;

// using no referenced names
cursor.next.prev = cursor.prev;
cursor.prev.next = cursor.next;
cursor = cursor.next;
manyNodes--;

// multiple assignments per line (most concise)
cursor.next.prev = cursor.prev;
cursor = cursor.prev.next = cursor.next;
manyNodes--;
```