

CSE 143

Computer Programming II

Linked Lists II



What Are We Doing...?

We're building an alternative data structure to an `ArrayList` with different efficiencies.

Today's Main Goals:

- Get more familiarity with `LinkedLists`
- Write more `LinkedList` methods
- Learn how to “protect” against `NullPointerExceptions`

Outline

- 1 Get more familiarity with changing `LinkedLists`
- 2 Write more methods in the `LinkedList` class
- 3 Protecting Against `NullPointerExceptions`

New Constructor

Create a constructor

```
public LinkedList(int n)
```

which creates the following `LinkedList`, when given n :



What kind of loop should we use?

A `for` loop, because we have numbers we want to put in the list.

What cases should we worry about?

We're creating the list; so, there aren't really "cases".

First Attempt

```

1 public LinkedList(int n) {
2
3
4     ListNode current = this.front;
5
6     for (int i = 1; i <= n; i++) {
7         current = new ListNode(i);
8
9         current = current.next;
10
11     }
12 }

```

/* Current State */

front



front

current



front current



front



Remember, to edit a LinkedList, we **MUST** edit one of the following:

- front, or
- node.next (for some ListNode node)

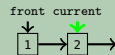
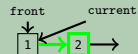
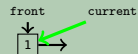
In our code above, we edit current, which is neither.

Second Attempt

```
1 public LinkedList(int n) {  
2  
3     if (n > 0) {  
4         //n is at least 1...  
5         this.front = new ListNode(1);  
6  
7         ListNode current = this.front;  
8  
9         for (int i = 1; i <= n; i++) {  
10            current.next = new ListNode(i);  
11  
12            current = current.next;  
13  
14        }  
15    }  
16 }
```

/* Current State */

front
↓



A New LinkedList Constructor: Another Solution

This other solution works by going backwards. Before, we were editing the next fields. Here, we edit the front field instead:

Different Solution!

```
1 public LinkedList(int n) {
2
3     for (int i = n; i > 0; i--) {
4         ListNode next = this.front;
5
6         this.front = new ListNode(i, next);
7
8     } /* Second time through the loop (for demo)... */
9         //ListNode next = this.front;
10
11         //this.front = new ListNode(i, next);
12
13 }
```

/ Current State */*

front
↓

front next
↓ ↓
n n

front next
↓ ↓
n n

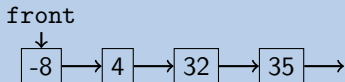
front next
↓ ↓
n-1 n

Outline

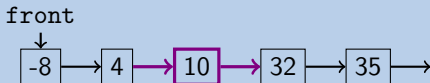
- 1 Get more familiarity with changing `LinkedLists`
- 2 Write more methods in the `LinkedList` class
- 3 Protecting Against `NullPointerExceptions`

addSorted

Write a method `addSorted(int value)` that adds `value` to a sorted `LinkedList` and **keeps it sorted**. For example, if we call `addSorted(10)` on the following `LinkedList`,



We would get:

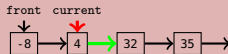
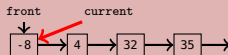
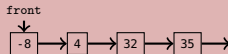


As always, we should approach this by considering the separate cases (and then drawing pictures):

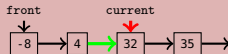
- We're supposed to insert at the front
- We're supposed to insert in the middle
- We're supposed to insert at the back

An Incorrect Solution

```
1 public void addSorted(int value) { //Say value = 10...
2
3     ListNode current = this.front;
4
5     while (current.data < value) {
6         current = current.next;
7
8     }
9
10
11 }
```



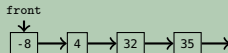
...the while loop continues...



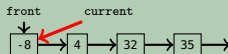
Uh Oh! We went too far! We needed the `next` field BEFORE us.

Fixing the Problem

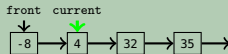
```
1 public void addSorted(int value) { //Say value = 10...
```



```
2     ListNode current = this.front;
```



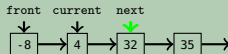
```
4 while (current.next.data < value) {
5     current = current.next;
```



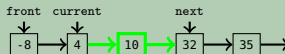
```
7 }
8
```

...the while loop STOPS now...

```
9     ListNode next = current.next;
```



```
11    current.next = new ListNode(value, next);
```

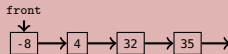


```
13 }
14 }
```

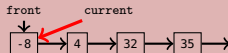
Does this cover all the cases?

Adding At The End?

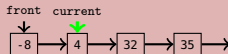
```
1 public void addSorted(int value) { //Say value = 40...
```



```
2     ListNode current = this.front;
```

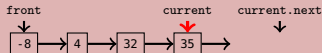


```
4 while (current.next.data < value) {
5     current = current.next;
```



```
7 }
8
9
```

...the while loop continues...



...AND IT KEEPS ON GOING...

current.next.data → NullPointerException!!!

```
10
11
12
13 }
```

We fell off the end of the LinkedList.
Idea: Make sure `current.next` exists.

Adding At The End?

```
public void addSorted(int value) {
    ListNode current = this.front;
    /* If we are making a check for current.next, we must
     * be sure that current is not null. */
    while (current.next.data < value) {
        /* Since we want to keep on going here,
         * the check must be made in the while loop.
         */
        current = current.next;
    }
}
```

A Fix?

```
public void addSorted(int value) {
    ListNode current = this.front;
    /* The extra check here is useless...we've already checked
     * current.next by the time we get to it. */
    while (current.next.data < value && current.next != null) {
        current = current.next;
    }
}
```

A Real Fix!

```
public void addSorted(int value) {
    ListNode current = this.front;
    while (current.next != null && current.next.data < value) {
        current = current.next;
    }
}
```

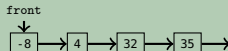
Our current code only sets `current` to a new `ListNode`. Importantly, this never updates `front`; so, we lose the new node.

Adding At The Beginning?

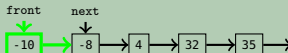
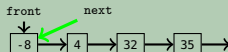
```

1 public void addSorted(int value) { //Say value = -10...
2
3     if (value < front.data) {
4         ListNode next = front;
5
6         front = new ListNode(value, next);
7
8     }
9     else {
10        ...
11    }
12 }

```



$-10 < -8 \rightarrow \text{true}$



Have we covered all of our cases now?

Outline

- 1 Get more familiarity with changing `LinkedLists`
- 2 Write more methods in the `LinkedList` class
- 3 Protecting Against `NullPointerExceptions`

With `LinkedList` code, every time we make a test (`if`, `while`, etc.), we need to make sure we're protected. Our current code is:

Working Code?

```
1 public void addSorted(int value) {
2     if (value < front.data) {
3         ListNode next = front;
4         front = new ListNode(value, next);
5     }
6     else {
7         while (current.next != null && current.next.data < value) {
8             current = current.next;
9         }
10
11         ListNode next = current.next;
12         current.next = new ListNode(value, next);
13     }
14 }
```

We're "protected" if we **know** we won't get a `NullPointerException` when trying the test. So, consider our tests:

- `value < front.data`
- `current.next != null && current.next.data < value`

So, Are We Protected?

Nope! What happens if `front == null`? We try to get the value of `front.data`, and get a `NullPointerException`. The fix:

Working Code!

```
1 public void addSorted(int value) {
2     if (front == null || value < front.data) {
3         ListNode next = front;
4         front = new ListNode(value, next);
5     }
6     else {
7         while (current.next != null && current.next.data < value) {
8             current = current.next;
9         }
10
11         ListNode next = current.next;
12         current.next = new ListNode(value, next);
13     }
14 }
```

Helpfully, this fix actually handles the empty list case correctly!

- Make sure to try all the cases:
 - Empty List
 - Front of Non-empty List
 - Middle of Non-empty List
 - Back of Non-empty List
- To Edit a LinkedList, the **assignment** must look like:
 - `this.front = <something>;`, or
 - `node.next = <something>;` (for some `ListNode node` in the list)
- Protect All Of Your Conditionals! Make sure that nothing can accidentally be `null`.
- When protecting your conditionals, make sure the less complicated check goes first.