

1 Cardinal Directions (Lecture 7)

Draw the box-and-pointer diagram that results from running the following code. A **DLLStringNode** is similar to a **Node** in a **DLList**. It has 3 instance variables: **prev**, **s**, and **next**. You may write each String inside the **s** box rather than drawing an arrow from **s** to a String.

```
public class DLLStringNode {
    DLLStringNode prev;
    String s;
    DLLStringNode next;

    public DLLStringNode(DLLStringNode prev, String s, DLLStringNode next) {
        this.prev = prev;
        this.s = s;
        this.next = next;
    }

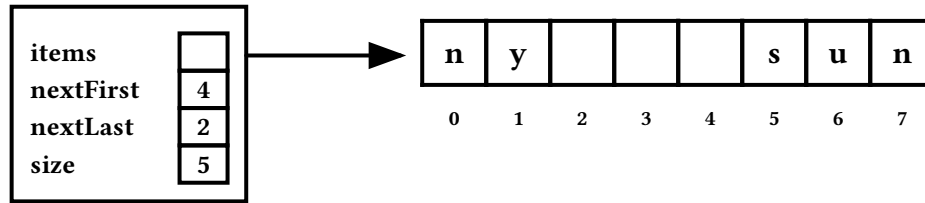
    public static void main(String[] args) {
        DLLStringNode L = new DLLStringNode(null, "eat", null);
        L = new DLLStringNode(null, "bananas", L);
        L = new DLLStringNode(null, "never", L);
        L = new DLLStringNode(null, "sometimes", L);
        DLLStringNode M = L.next;
        DLLStringNode R = new DLLStringNode(null, "shredded", null);
        R = new DLLStringNode(null, "wheat", R);
        R.next.next = R;
        M.next.next.next = R.next;
        L.next.next = L.next.next.next;

        /* Optional practice below. */

        L = M.next;
        M.next.next.prev = R;
        L.prev = M;
        L.next.prev = L;
        R.prev = L.next.next;
    }
}
```

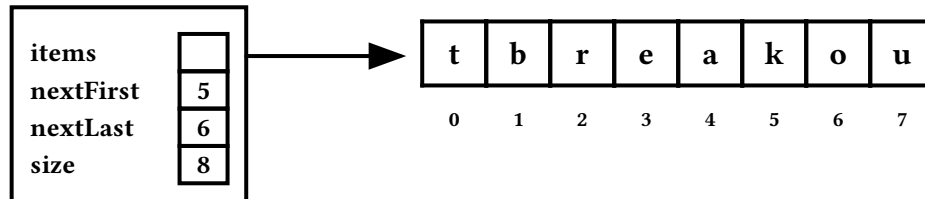
2 In Circles (Lecture 8)

Consider an `ArrayDeque` with state shown below.



- (a) What abstract list of elements does this data structure represent? Write them out from first to last. How do you know, i.e. what invariant do you have in mind?

Consider a completely new `ArrayDeque`. It's looking pretty cozy in that backing array!



- (b) What list of elements does this new object represent? Write them out from first to last. Note: If your invariant from part (a) was well written, this should help you avoid mistakes.

- (c) Draw what the backing array would look like after calling `addLast("s")` on this data structure. What would `nextFirst`, `nextLast`, and `size` be equal to? There is more than one correct solution.

Assume this data structure uses a scaling factor of 2.

3 Off to the Races (Lecture 7)

Consider the following Lists, using the **SLList**, **DLList**, and **AList** classes as implemented in lecture. As a reminder,

- **SLList** is a singly-linked list with a sentinel.
- **DLList** is a circular doubly-linked lists with a sentinel.
- **AList** is an array list with dynamic resizing.

```
public SLList clementineSurprise = new SLList();
public DLList superstitiousRealism = new DLList();
public AList jovialMerrymen = new AList();
```

Assume that each list contains 1000 items. For each set of blocks of code below, circle the one that you'd expect to run **faster** and briefly explain why.

(a) <pre>for (int i = 0; i < 100; i += 1) { clementineSurprise.addLast(); }</pre>	<pre>for (int i = 0; i < 200; i += 1) { superstitiousRealism.addLast(); }</pre>
--	--

(b) <pre>for (int i = 0; i < 900; i += 1) { jovialMerrymen.removeLast(); }</pre>	<pre>for (int i = 0; i < 200; i += 1) { clementineSurprise.removeLast(); }</pre>
---	---

(c) <pre>for (int i = 0; i < 300; i += 1) { superstitiousRealism.get(i); }</pre>	<pre>for (int i = 0; i < 300; i += 1) { jovialMerrymen.get(i); }</pre>
---	---

- (d) Imagine that we add a **last** pointer to our **SLList** class that keeps track of the last element of the list. We then modify **addLast** and **removeLast** to access **last** and update it accordingly. Which of your answers, if any, will change, and why?

4 Remove Duplicates (Lecture 7)

Using the simplified **DLList** class defined below, implement the `removeDuplicates` method. `removeDuplicates` should remove all duplicate items from the **DLList**. For example, if our initial list is [8, 4, 4, 6, 4, 10, 12, 12], our final list should be [8, 4, 6, 10, 12]. You may **not** assume that duplicate items are grouped together, or that the list is sorted! Do not worry about efficiency.

```
public class DLList {
    Node sentinel;
    public DLList() { // Constructor not shown... }
    public class Node { int item; Node prev; Node next; }
    public void removeDuplicates() {

        Node ref = _____;
        Node checker;
        while (_____) {
            checker = _____;
            while (_____) {
                if (_____) {
                    Node checkerPrev = checker.prev;
                    Node checkerNext = checker.next;

                    _____;
                    _____;
                }
                checker = _____;
            }
            ref = _____;
        }
    }
}
```