

1 The Mystery of the Walrus

- (a) Consider the code below. Next to each blank, write down the expected output. Alternatively, if it's impossible to predict the output, write "unknown".

Implementations for `obliterate`, `IntSquasher`, `shamble`, and `agglutinate` are unknown.

```
public class Walrus {
    public static void main(String[] args) {
        int x = 10;
        obliterate(x);

        System.out.println(x);
        int y = 20;
        IntSquasher isq = new IntSquasher(y);

        System.out.println(y);

        int[] x = new int[]{1, 2, 3};
        shamble(x[0]);

        System.out.println(x[0]);
        agglutinate(x);

        System.out.println(x[1]);
    }
}
```

10

20

1

unknown

- (b) Consider the class `MyInteger` below.

```
public class MyInteger {
    public int val;
    public MyInteger(int val) {
        this.val = val;
    }

    @Override
    public String toString() {
        return String.valueOf(this.val);
    }
}
```

If `x` was instantiated as

```
MyInteger[] x = new MyInteger[] {MyInteger(1), MyInteger(2), MyInteger(3)};
```

Would any of your answers change? If so, which ones, and why? If not, why not?

`int` is not a reference type. `MyInteger` is! When you pass `x[0]` into `agglutinate`, it could access `x[0].val` and modify it, changing the value of `x[0]` in `main`'s scope. As such, `System.out.println(x[0])` now prints an unknown value!

- (c) Implementations for `invertify`, `scrub`, and `feed` are unknown.

```

public class WalrusReview {
    public int v;
    public static String name;

    public WalrusReview(int v) {
        this.v = v;
        name = "Scott";
        v = -10;
    }

    public static void main(String[] args) {
        int z = 10;
        WalrusReview wr = new WalrusReview(z);

        System.out.println(z); 10

        System.out.println(wr.v); 10

        invertify(wr.v);

        System.out.println(wr.v); 10

        scrub(WalrusReview.name);

        System.out.println(WalrusReview.name); Scott

        z = 10;
        wr = new WalrusReview(z);
        feed(WalrusReview);

        System.out.println(z); 10

        System.out.println(wr.v); unknown

        System.out.println(WalrusReview.name); unknown
    }
}

```

2 Cardinal Directions

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**.

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

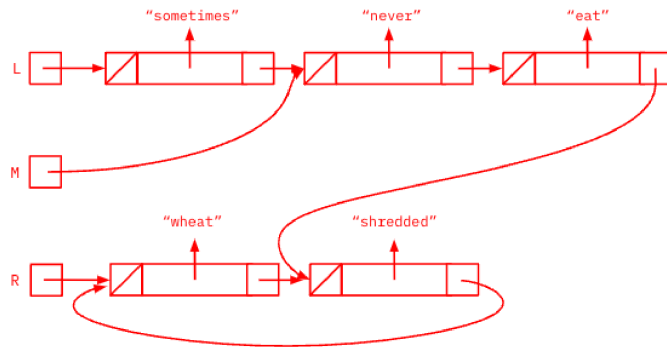
    public DLLStringNode(DLLStringNode prev, String s, DLLStringNode next) {
        this.prev = prev;
        this.s = s;
        this.next = next;
    }

    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;

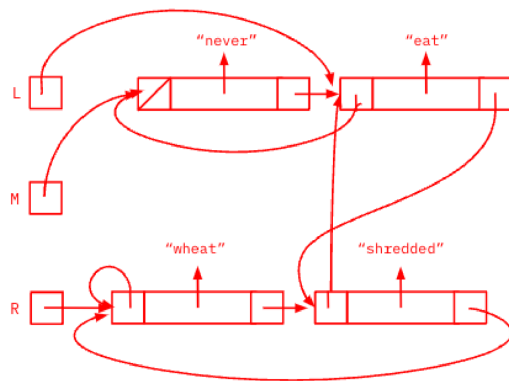
        /* Optional practice below. */

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

Before optional practice:



After optional practice:



3 Gridify

- (a) Consider a circular sentinel implementation of an **SLList** of **Nodes**. For the first **rows** * **cols** **Nodes**, place the item of each **Node** into a 2D **rows** × **cols** array in row-major order. Elements are sequentially added filling up an entire row before moving onto the next row.

For example, if the **SLList** contains elements $5 \rightarrow 3 \rightarrow 7 \rightarrow 2 \rightarrow 8$ and **rows** = 2 and **cols** = 3, calling **gridify** on it should return this grid.

5	3	7
2	8	0

If the SLList contains fewer elements than the capacity of the 2D array, the remaining array elements should be 0; if it contains more elements, ignore the extra elements.

```
public class SLList {
    Node sentinel;

    public SLList() {
        this.sentinel = new Node();
    }

    private static class Node {
        int item;
        Node next;
    }

    public int[][] gridify(int rows, int cols) {
        int[][] grid = new int[rows][cols];;

        gridifyHelper(grid, sentinel.next, 0);
        return grid;
    }

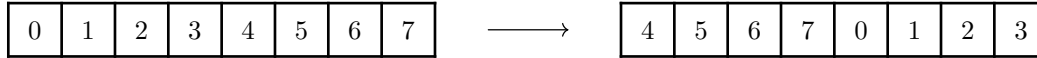
    private void gridifyHelper(int[][] grid, Node curr, int numFilled) {
        if (curr == sentinel || numFilled >= grid.length * grid[0].length) {
            return;
        }

        int row = numFilled / grid[0].length;
        int col = numFilled % grid[0].length;

        grid[row][col] = curr.item;
        gridifyHelper(grid, curr.next, numFilled + 1);
    }
}
```

4 Rotate (Extra)

Write a function that, when given an array **A** and integer **k**, returns a new array whose contents have been shifted **k** positions to the right, wrapping back around to index 0 if necessary. For example, if **A** contains the values 0 through 7 inclusive and **k** = 12, then the array returned after calling **rotate(A, k)** is shown below on the right:



k can be arbitrarily large or small - that is, **k** can be a positive or negative number. If **k** is negative, shift **k** positions to the left. After calling **rotate**, **A** should remain unchanged. (This means that rotate is **nondestructive!**)

Hint: you may find the modulo operator % useful. Note that the modulo of a negative number is still negative (i.e. $(-11) \% 8 = -3$).

```
/** Returns a new array containing the elements of A shifted k positions to the right. */
public static int[] rotate(int[] A, int k) {
    int rightShift = k % A.length;

    if (rightShift < 0) {
        rightShift += A.length;
    }

    int[] newArr = new int[A.length];

    for (int i = 0; i < A.length; i++) {
        int newIndex = (i + rightShift) % A.length;

        newArr[newIndex] = A[i];
    }
    return newArr;
}
```