CS 61B Fall 2025

Scope, Static, Linked Lists, Arrays

Discussion 02: September 8, 2025

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]);                 _____

    }
}
```

(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?

(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);                          _____

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

        invertify(wr.v);

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

        scrub(WalrusReview.name);

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

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

        System.out.println(z);                          _____

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

        System.out.println(WalrusReview.name);          _____
    }
}
```

This page was intentionally left blank.

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;
  }
}
```

3 Gridify

(a) Consider a circular sentinel implementation of an **SLList** of **Node**s. For the first **rows** * **cols** **Node**s, 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 $5 \to 3 \to 7 \to 2 \to 8$ and **rows** $= 2$ and **cols** $= 3$, calling **gridify** on it should return this grid.

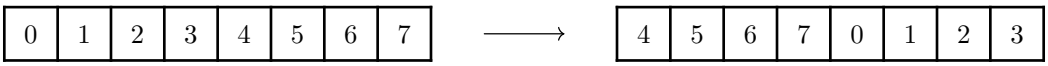
5	3	7
2	8	0

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 = _____;

    _____;

    return grid;
  }

  private void gridifyHelper(int[][] grid, Node curr, int numFilled) {

    if (_____) {

      return;
    }


    int row = _____;

    int col = _____;


    grid[row][col] = _____;

    _____;

  }
}
```

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 = _____;

    if (_____) {

        _____;
    }


    int[] newArr = _____;

    for (_____) {

        int newIndex = _____;

        _____;
    }
    return newArr;
}
```