Topical Review Session
Section 1: Pointers, LinkedLists, Arrays

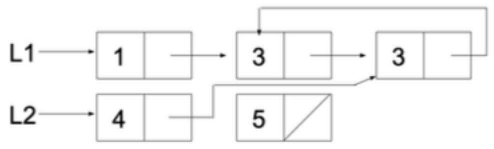
MT1 Review

1 Pointers

1. Draw the resulting box and pointer diagrams for the following lines of code. The head of an IntList is its value, and the tail is a pointer to the next node in the list.

```
IntList L1 = IntList.list(1, 2, 3);
IntList L2 = IntList.list(4, 5);
L1.tail.head = 3;
L2.tail = L1.tail.tail;
L2.tail.tail = L1.tail;
```

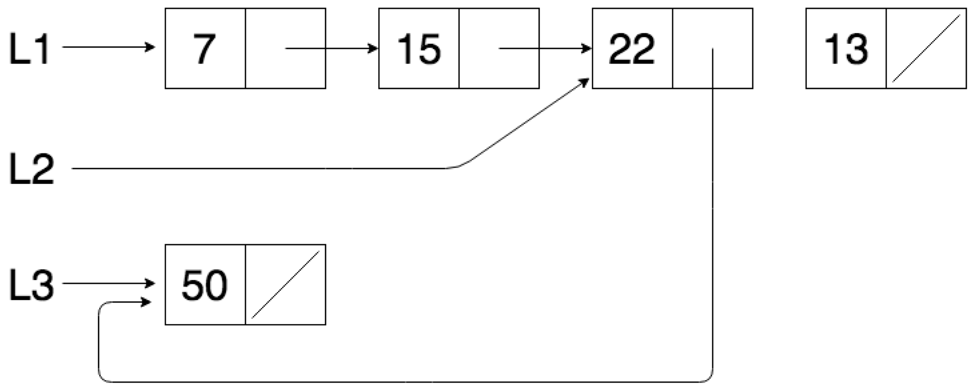
Solution:



2. Draw the resulting box and pointer diagrams for the following lines of code.

```
IntList L1 = IntList.list(7,15,22,31);
IntList L2 = L1.tail.tail;
L2.tail.head = 13;
L1.tail.tail.tail = L2;
IntList L3 = IntList.list(50);
L2.tail.tail = L3;
```

Solution:



3. What would the output of the following lines of code be? Be sure to draw a box-and-pointer diagram!

```
public static void main(String[] args) {
    IntList L1 = IntList.list(8, 3, 6, 4);
    IntList L2 = IntList.list(4, 5, 9, 0);
    IntList L3 = L2;
    int x = 4;
    mystery(L1, L3, x);
    System.out.println(L1);
    System.out.println(L2);
    System.out.println(x);
}


public static void mystery(IntList L1, IntList L2, int x) {
    L1.head = 23;
    L2.tail.tail = L1.tail;
    L1.tail.tail.head = L2.tail.head;
    x += 16;
    L2 = IntList.list(1, 2);
}
```

Solution:

[23, 3, 5, 4]
[4, 5, 3, 5, 4]
4

4. Let's say a method has the following signature: "public int foo(int x)". What is stored in the variable x?
The value 4 is stored in x, since x is of primitive int type.

5. Similarly, let's say some other method has the following signature: "public boolean boo(IntList y)". What is stored in the variable y? What happens if we change the value of y in boo?
   Since y is of type IntList (an object), it stores a pointer to an IntList (in other words, it stores the address where the IntList is located in memory). When this pointer is changed in boo, the original IntList that y pointed to will remain unchanged.

2 Arrays

Describe what each of the following methods do. You may assume that `values` contains at least one element.

```
private static boolean method1 (int[] values) {
    int k = 0;
    while (k < values.length - 1) {
        if (values[k] > values[k+1]) {
            return false;
        }
        k = k + 1;
    }
    return true;
}
```

Solution: `method1` returns true if `values` is non-decreasing, i.e. if each value in `values` is larger than *or equal to* the previous element.

```
private static void method2 (int[] values) {
    int k = 0;
    while (k < values.length / 2) {
        int temp = values[k];
        values[k] = values[values.length - 1 - k];
        values[values.length - 1 - k] = temp;
        k = k + 1;
    }
}
```

Solution: `method2` reverses `values` in place. Note that method2 has no return value and instead mutates `values`.

3 Linked Lists

7. Consider the following:

```
public class Point {
    public int x;
    public int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}


public class PointList {

    private class Stuff {

        Point item;
        Stuff next;
        Stuff prev;

        Stuff(Point item, Stuff next, Stuff prev) {
        this.item = item;
        this.next = next;
        this.prev = prev;
        }
    }

    public int size;
    private final Stuff sentinel;

    public PointList() {
    // implementation omitted
    }

    public void addMiddle(Point p) {

    // your answer here

    }
}
```

The method `addMiddle(Point p)` is supposed to add some `Point p` to the middle of `this` instance of `PointList`. Assuming that the size of the list is greater than 1, write the code that would properly implement `addMiddle(Point p)`.

Answer:

```
Stuff midStuff = this.sentinel;
for (int i = 0; i < this.size / 2; i += 1) {
    midStuff = this.sentinel.next;
}

Stuff newStuff = new Stuff(p, midStuff.next, midStuff)
midStuff.next.prev = newStuff;
midStuff.next = newStuff;
this.size += 1;
```