

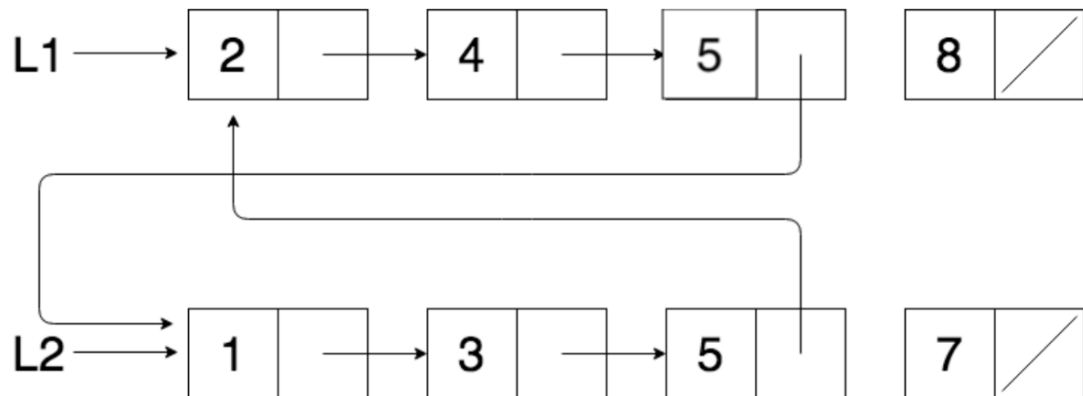
1 Pointer Practice

Draw the resulting box and pointer diagram for the L1 Singly Linked IntList after the following code is executed:

1. IntLists

```
IntList L1 = IntList.list(2, 4, 6, 8);
IntList L2 = IntList.list(1, 3, 5, 7);
L1.tail.tail.head = 5;
L2.tail.tail.tail = L1;
L1.tail.tail.tail = L2;
```

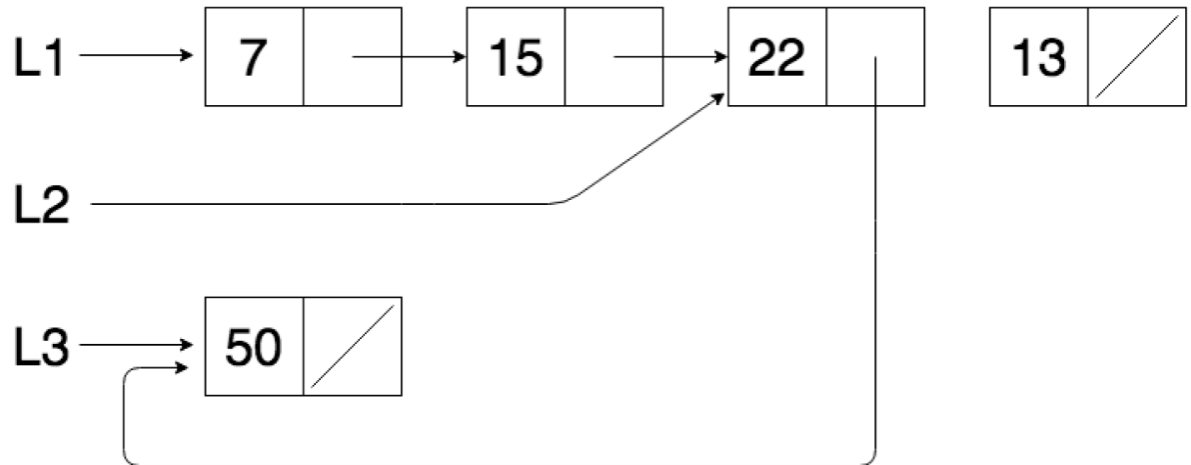
Solution:



2. IntLists

```
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:



2 Destructivity

Will is working on his app, CalTransit (check it out on the Apple App Store!) and is writing a function that given an `IntList`, appends the length of the `IntList` at the end of the list. Tiger thinks writing a non destructive function will be a better idea. Tiger writes the following method:

```
public static IntList addLength(IntList i):  
    temp = i  
    temp.addLast(i.length)  
    return temp
```

Assuming that the `IntList` class was already correctly defined, will this method execute as expected? If not, how can it be fixed?

Solution: The method will not work as expected because it is destructive. By assigning `temp = i`, we simply have `temp` point to the `IntList` that `i` is pointing to rather than making a deep copy of `i`. Thus, mutating `temp` will also mutate `i` since they are pointing to the same list. In order to overcome this we would need to make a deep copy of `i` and store that `IntList` in `temp`.

3 Skipping Stones

Write a function that takes in an `IntList L`, which must contain at least one element, and returns an `IntList` with every odd indexed element removed. Try out both the destructive and nondestructive approaches.

1. Nondestructive

Solution:

```
public static IntList skipNondestructive (IntList L) {  
    IntList pointer = new IntList(L.head);;  
    IntList result = pointer;
```

```

        while (L.tail != null && L.tail.tail != null) {
            L = L.tail.tail;
            pointer.tail = new IntList(L.head);
            pointer = pointer.tail;
        }
    }
    return result;
}

```

2. Destructive

Solution:

```

public static IntList skipDestructive (IntList L) {
    IntList pointer = L;
    while (pointer.tail != null && pointer.tail.tail != null) {
        pointer.tail = pointer.tail.tail ;
        pointer = pointer.tail;
    }
    pointer.tail = null
    return L;
}

```