### Boxes and Pointers

Draw a box and pointer diagram to represent the IntLists L, M, and N after each statement.

```java
IntList L = IntList.list(1, 2, 3, 4);
IntList M = L.tail.tail;
IntList N = IntList.list(5, 6, 7);
N.tail.tail.tail = N;
L.tail.tail = N.tail.tail.tail.tail;
M.tail.tail = L;
```

In lecture, we discussed Consumers. If you are familiar with Python, Consumers are analogous to lambda functions. Draw a box and pointer diagram to represent the IntLists A and B after each statement.

```java
Consumer<IntList> trunc = (list) -> {
    int x = list.head;
    for (int i = 0; i < x && list.tail != null; i++) {
        list = list.tail;
    }
    list.tail = null;
};

IntList A = IntList.list(2, 5, 3, 1, 2, 4, 8);
IntList B = A.tail.tail.tail;
trunc.accept(B);
trunc.accept(A);
```
Extra: Draw a box and pointer diagram to represent the IntLists L1, L2, and L3 after each statement.

```java
IntList L1 = IntList.list(1, 2, 3);
IntList L2 = new IntList(4, L1.tail);
L2.tail.head = 13;
L1.tail.tail.tail = L2;
IntList L3 = IntList.list(50);
L2.tail.tail = L3;
```
### Destructive or Nondestructive?

Below is a method that takes in an IntList and returns the value of the head of the IntList. Assume that L is never null.

```java
/** Returns the head of IntList L. Assumes that L is not null. */
public static int getHead(IntList L) {
    int listHead = L.head;
    L = new IntList(5, null);
    return listHead;
}
```

Is the above method destructive or nondestructive? Explain.
3 Reversing a Linked List

Implement the following method, which reverses an IntList nondestructively. The original IntList should not be modified. Instead, the method should return a new IntList that contains the elements of L in reverse order.

```java
/** Nondestructively reverses IntList L. */
public static IntList reverseNondestructive(IntList L) {

}
```

Extra: Implement the following method which destructively reverses an IntList.

```java
/** Destructively reverses IntList L. */
public static IntList reverseDestructive(IntList L) {

}
```
4 Inserting into a Linked List

Implement the following method to insert an element item at a given position position of an IntList L. For example, if L is (1 -> 2 -> 4) then the result of calling insert(L, 3, 2) yields the list (1 -> 2 -> 3 -> 4). This method should modify the original list (do not create an entirely new list from scratch). This method should return a pointer to the front of the resulting IntList. Use recursion.

/** Inserts item at the given position in IntList L and returns the resulting * IntList. If the value of position is past the end of the list, inserts the * item at the end of the list. Uses recursion. */
public static IntList insertRecursive(IntList L, int item, int position) {

Extra: Implement the method described above using iteration. insertIterative is a destructive method and should therefore modify the original list (just like the previous problem, do not create an entirely new list from scratch).

/** Inserts item at the given position in IntList L and returns the resulting * IntList. If the value of position is past the end of the list, inserts the * item at the end of the list. Uses iteration. */
public static IntList insertIterative(IntList L, int item, int position) {

}
Extra: Shifting a Linked List

Implement the following method to circularly shift an IntList to the left by one position destructively. For example, if the original list is (5 -> 4 -> 9 -> 1 -> 2 -> 3) then this method should return the list (4 -> 9 -> 1 -> 2 -> 3 -> 5). Because it is a destructive method, the original IntList should be modified. Do not use the word new.

/** Destructively shifts the elements of the given IntList L to the
 * left by one position. Returns the first node in the shifted list. */
public static IntList shiftListDestructive(IntList L) {

}