CS 61B Spring 2026

Testing, IntLists, SLLists

Discussion 03: February 02, 2026

1 Boxes and Pointers

Draw a box and pointer diagram to represent the IntLists `L1`, `L2`, and `L3` after each statement.

```
IntList L1 = IntList.list(1, 2, 3);
IntList L2 = new IntList(4, L1.rest);
L2.rest.first = 13;
L1.rest.rest.rest = L2;
IntList L3 = IntList.list(50);
L2.rest.rest = L3;
```

2 Replace

(a) Fill in the `IntList` method `static void replace(IntList L, int a, int b)` below. Your implementation should be destructive (modifies the list in place) and iterative (does not use recursion).

```
public class IntList {
    public int first;
    public IntList rest;

    /** Replaces all instances of a with b in L.
     *  Modfies the passed list. Non-recursive.
     */
    public static void replace(IntList L, int a, int b) {

        IntList p = _____;

        while (_____) {

            _____

            _____

            _____

            _____

        }
    }
}
```

(b) Fill in the method **`static IntList replace(IntList L, int a, int b)`** below. Your implementation should be non-destructive (does not modify the invoked list) and recursive.

```
public class IntList {
    public int first;
    public IntList rest;

    /** Returns a copy of the provided list but with all occurrences
     * of a replaced by b. Does not modify the current list. Recursive.
     */
    public static IntList replace(IntList L, int a, int b) {

        if (_____________________________________________________________) {
            return null;
        }


        if (_____________________________ == _____________________________) {

            return new IntList(_____________________________________________;
        } else {

            _____________________________________________________________;
        }
    }
}
```

(c) Lastly, write a *destructive*, recursive version of **`replace`** below:

```
/** Replaces all occurrences of a with b. Modifies the current list. Recursive. */
public static void replace(IntList L, int a, int b) {

    if (_____________________________________________________________) {

        _____________________________________________________________;
    }


    if (_____________________________________________________________) {

        _____________________________________________________________;
    }


    _____________________________________________________________;
}
```

3 Deduplication

Fill in the blanks to implement the **removeDuplicates** method correctly. Given a sorted linked list of items, remove the duplicates. For example, given `1 -> 2 -> 2 -> 2 -> 3`, mutate it to become `1 -> 2 -> 3` destructively. Your implementation should be destructive and non-recursive.

```
public class IntList {
    public int first;
    public IntList rest;

    public static void removeDuplicates(IntList p) {

        if (_____) {
            return;
        }

        IntList curr = _____;

        IntList prev = _____;

        while (_____) {

            if (curr.first == _____) {

                _____;
            } else {

                _____;
            }

            _____;
        }
    }
}
```

Reminder: Only write one statement per line.

4 SLList

Implement the **`SLList.insert`** method which takes in an **`int x`** and an **`int position`**. It inserts **`x`** at the given **`position`**.

For example:

- If the SLList is $5 \to 6 \to 2$, then **`insert(10, 1)`** results in $5 \to 10 \to 6 \to 2$.
- If the SLList is $5 \to 6 \to 2$, **`insert(10, 7)`** results in $5 \to 6 \to 2 \to 10$.

Assume that **`position`** is a non-negative integer. For an easier challenge, assume **`position`** is between **`0`** and **`this.size() + 1`**. For a bigger challenge, if **`position`** is less than **`0`** or greater than **`this.size()`**, then insert at the leftmost or rightmost end, respectively.

```
public class SLList {
  private class IntNode {
    public int item;
    public IntNode next;
    public IntNode(int item, IntNode next) {
      this.item = item;
      this.next = next;
    }
  }

  private IntNode sentinel;
  public int size;

  public void addFirst(int x) {
      sentinel.next = new IntNode(x, sentinel.next);
      size += 1;
  }

  public void insert(int x, int position) {
```

```
  }
}
```