

CSM Berkeley 61B, Spring 2015: Week 3 Solutions

1. What's wrong with these sum functions?

File: `BuggySums.java`

```
class BuggySums {  
  
    public static int buggySum1(int[] a) {  
        int total = 0;  
        for (int i = 1; i < a.length; i++) {  
            total += a[i];  
        }  
        return total;  
    }  
}
```

`i` starts at 1, and so the 0th iteration is skipped.

```
    public static int buggySum2(int[] a) {  
        int total = 0;  
        for (int i = 0; i <= a.length; i++) {  
            total += a[i];  
        }  
        return total;  
    }  
}
```

`i` goes up to `a.length`, but the last item of `a` is at `a[a.length - 1]`.

```
    public static int buggySum3(int[] a) {  
        int i = 0;  
        int total = 0;  
        while (i < a.length) {  
            total += a[i];  
        }  
        return total;  
    }  
}
```

`i` is never incremented, so this is stuck in an infinite loop.

```
}
```

2a. Write middle, which takes in array of ints and returns the middle element.

If no element is in the exact middle, return the one to the left of the middle.
Don't over-think this! :)

File: `ArrayExample.java`

```
import java.util.Arrays;

class ArrayExample {
    public static int middle(int[] arr) {
```

```
        return arr[(arr.length - 1)/2];
```

```
    }
```

2b. Write reverse, which takes in an array of ints and reverses its elements in-place.

```
    public static void reverse(int[] arr) {
```

```
        int len = arr.length;
        for (int i = 0; i < len/2; i++) {
            int temp = arr[i];
            arr[i] = arr[len - i - 1];
            arr[len - i - 1] = temp;
        }
```

```
    }
```

```
    public static void main(String[] args) {
        int[] test1 = {1, 3, 3, 7, 42};
        System.out.println(Arrays.toString(test1));
        reverse(test1);
        System.out.println(Arrays.toString(test1));
    }
}
```

3. Write middle, for SLists.

Hint: why are our pointers called slow and fast?

If no element is in the exact middle, return the one to the left of the middle.

File: `SList.java`

```
public class SList {
    private SListNode head;
    public SList(SListNode head) {
        this.head = head;
    }
    public SList() {
        this(null);
    }

    public static Object middle(SList list) {

        SListNode slow = list.head;
        SListNode fast = list.head;

        while (fast != null && fast.next != null && fast.next.next != null) {

            slow = slow.next;
            fast = fast.next.next;

        }
        return slow.item;

    }

    public String toString() {
        String result = "";
        for (SListNode cur = head; cur != null; cur = cur.next)
            result += cur.item.toString() + " ";
        return result;
    }

    public static void main(String[] args) {
        SList l = new SList(new SListNode(1, new SListNode(2, new SListNode(3))));
        System.out.println("l = " + l);
        System.out.println("l middle: " + middle(l));
    }
}

class SListNode {
    Object item; SListNode next;
    SListNode(Object item, SListNode next) {
        this.item = item; this.next = next;
    }
    SListNode(Object item) {
        this(item, null);
    }
}
```

4. Spot the bug! (extra time)

File: `IntListBug.java`

```
class IntListBug {

    /**
     * Returns a list consisting of the elements of A followed by the
     * elements of B. May NOT modify items of A.
     */
    public static IntList buggyCatenate(IntList A, IntList B) {
        IntList C = new IntList(A.head, A.tail);
```

This step attempts to copy A to C, but actually only copies over the first node of A. The rest, `A.tail`, contains the same nodes as A. The way to fix this would be to use a loop to copy over *all* the nodes in A. For example:

```
        IntList Apointer = A.tail, Cpointer = C;
        while (Apointer != null) {
            Cpointer.tail = new IntList(Apointer.head, null);
            Apointer = Apointer.tail;
            Cpointer = Cpointer.tail;
        }
        // Old code, no longer needed:
        // IntList list2 = C;
        // while (list2.tail != null) {
        //     list2 = list2.tail;
        // }
        // list2.tail = B;
        Cpointer.tail = B;
```

```
    }

    return C;
}

public static void main(String[] args) {
    IntList A = IntList.list(1, 2, 3);
    IntList B = IntList.list(4, 5, 6);

    System.out.println(A);    // 1 2 3
    IntList C = buggyCatenate(A, B);
    System.out.println(C);    // 1 2 3 4 5 6  this seems to work
    System.out.println(A);    // 1 2 3 4 5 6  <-- fixed; now prints 1 2 3
}
}
```

File: `IntList.java`

```
public class IntList {
    public int head;
    public IntList tail;

    /** Constructs an IntList from a head int and tail IntList. */
    public IntList(int head, IntList tail) { ... }

    /** Constructs an IntList from the list of arguments. */
    public static IntList list(Integer... args) { ... }

    /** Returns string representation of the IntList. */
    public String toString() { ... }
}
```