CS 61B Discussion 4: Exam Prep Spring 2020

1 Give em the 'Ol Switcheroo

For each function call in the `main` method, write out the x and y values of both foobar and baz after executing that line. (Spring 2015, MT1)

```
public class Foo {
    public int x, y;

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

    public static void switcheroo (Foo a, Foo b) {
        Foo temp = a;
        a = b;
        b = temp;
    }

    public static void fliperoo (Foo a, Foo b) {
        Foo temp = new Foo(a.x, a.y);
        a.x = b.x;
        a.y = b.y;
        b.x = temp.x;
        b.y = temp.y;
    }

    public static void swaperoo (Foo a, Foo b) {
        Foo temp = a;
        a.x = b.x;
        a.y = b.y;
        b.x = temp.x;
        b.y = temp.y;
    }

    public static void main (String[] args) {
        Foo foobar = new Foo(10, 20);
        Foo baz = new Foo(30, 40);
        switcheroo(foobar, baz); foobar.x: 10 foobar.y: 20 baz.x: 30 baz.y: 40
        fliperoo(foobar, baz);   foobar.x: 30 foobar.y: 40 baz.x: 10 baz.y: 20
        swaperoo(foobar, baz);   foobar.x: 10 foobar.y: 20 baz.x: 10 baz.y: 20
    }
}
```

2 Flatten

Write a method `flatten` that takes in a 2-D array `x` and returns a 1-D array that contains all of the arrays in `x` concatenated together. (Summer 2016, MT1)

For example, `flatten({{1, 2, 3}, {}, {7, 8}})` should return {1, 2, 3, 7, 8}.

```
public static int[] flatten(int[][] x) {
    int totalLength = 0;

    for (int i = 0; i < x.length; i++) {
        totalLength += x[i].length;
    }

    int[] a = new int[totalLength];
    int aIndex = 0;

    for (int i = 0; i < x.length; i++) {
        for (int j = 0; j < x[i].length; j++) {
            a[aIndex] = x[i][j];
            aIndex++;
        }
    }

    return a;
}
```

Here is an alternate solution that uses `arraycopy`.

```
public static int[] flatten(int[][] x) {
      int totalLength = 0;

      for (int i = 0; i < x.length; i++) {
         totalLength += x[i].length;
      }

      int[] a = new int[totalLength];
      int aIndex = 0;

      for (int[] arr: x) {
         System.arraycopy(arr, 0, a, aIndex, arr.length);
         aIndex += arr.length;
      }

      return a;
   }
}
```

3 IntList to Array

For this problem we will implement a version of `arraycopy` that copies elements from an IntList into an array of `int`s. As a reminder, here is the `arraycopy` method:

```
System.arraycopy(Object src, int sourcePos, Object dest, int destPos, int len)
```

`System.arraycopy` copies `len` elements from array `src` (starting at index `source`) to array `destArr` (starting from index `dest`).

To simplify things, let's restrict ourselves to using only `int[]`, and assume that `srcList` and `destArr` are not null. Additionally, assume that `sourcePos`, `destPos`, and `len` will not cause an IndexOutOfBoundsException to be thrown.

For example, let IntList `L` be (1 -> 2 -> 3 -> 4 -> 5) and `int[] arr` be an empty array of length 3. Calling `arrayCopyFromIntList(L, 1, arr, 0, 3)` will result in `arr={2, 3, 4}`.

```
/** Works just like System.arraycopy, except srcList is of type IntList. */
public static void arrayCopyFromIntList(IntList srcList, int sourcePos,
        int[] destArr, int destPos, int len) {
    for (int i = 0; i != sourcePos; i += 1;) {
        srcList = srcList.tail;
    }

    for (int i = destPos ; i < destPos + len; i += 1) {
        destArr[i] = srcList.head;
        srcList = srcList.tail;
    }
}
```

4 Adding Long Numbers: IntList Adder

Here we'll write code that interprets IntLists as numbers by thinking of them as a list of digits. Then, we'll write a method to add two such numbers. This is useful for adding numbers with values much larger than what Java can represent using Integers.

For this question we will be representing the input and output numbers with the least significant digit (LSD) on the left and the most significant digit (MSD) on the right, which will have the effect of making them look backwards. For example, the number 12345 should be represented as an IntList with the items (5 -> 4 -> 3 -> 2-> 1).

Here is an example of adding 321 to 99:

Let IntList `x` be (1 -> 2 -> 3) and IntList `y` be (9 -> 9 -> 0). Calling `add(x, y)` will yield an IntList representing the sum (0 -> 2 -> 4), or 420.

Assume that you will not have to deal with negative numbers and that every element in the IntLists is an integer in the range [0, 9] (i.e. numbers between 0 and 9, inclusive). To further simplify things, assume that both `x` and `y` have the same length.

```
public IntList add(IntList x, IntList y) {
    int carry = 0;
    IntList sum = new IntList(-1, null);
    IntList last = sum;

    while (x != null && y != null) {
        int digit = x.head + y.head;
        digit += carry;
        carry = 0;

        if (digit > 9) {
            digit = digit - 10;
            carry = 1;
        }

        last.tail = new IntList(digit, null);
        last = last.tail;
        x = x.tail;
        y = y.tail;
    }

    if (carry == 1) {
        last.tail = new IntList(carry, null);
    }

    return sum.tail;
}
```