UNIVERSITY OF CALIFORNIA
Department of Electrical Engineering
and Computer Sciences
Computer Science Division

CS61B
Fall 2016

P. N. Hilfinger

Test #1 Solution

Reference Material.

```
/* arraycopy(FROM_ARR, FROM_INDEX, TO_ARR, TO_INDEX, LENGTH) */
import static java.lang.System.arraycopy;

public class IntList {
    /** First element of list. */
    public int head;
    /** Remaining elements of list. */
    public IntList tail;

    /** A List with head HEAD0 and tail TAIL0. */
    public IntList(int head0, IntList tail0)
    { head = head0; tail = tail0; }

    /** A List with null tail, and head = 0. */
    public IntList() { this(0, null); }

    /** Returns a new IntList containing the ints in ARGS. */
    public static IntList list(Integer ... args) {
           // Implementation not shown
    }

    /** Returns true iff L (together with the items reachable from it) is an
     *  IntList with the same items as this list in the same order. */
    @Override
    public boolean equals(Object L) {
           // Implementation not shown
    }

    /** Return the length of the non-circular list headed by L. */
    @Override
    public int size() {
           // Implementation not shown
    }

}
```

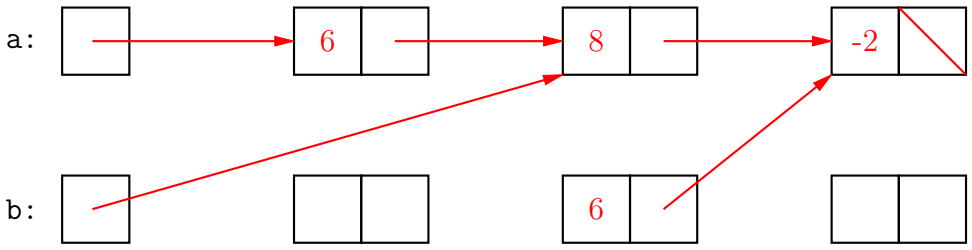
1. [3 points] For the following code snippets, fill in the box and pointer diagrams to show the variables and objects created and their contents, using the empty boxes provided. The double boxes represent `IntList` objects (see definition on page 2), with the left box containing the `head` field. Show also any output produced. You may not need all the boxes or output lines provided.

(a)
```
IntList a = IntList.list(6, 8, 7);
a.tail.tail.head = a.tail.head - 10;
IntList b =
   new IntList(a.head, a.tail.tail);
b = a.tail;

System.out.println(b.tail.head);

System.out.println(a.head);
```

Write Output Here:

-2

6



(b) The vertical stack of boxes represents an array object.

```
IntList a = IntList.list(65, 82, 77);
IntList[] b = new IntList[3];
b[1] = a;
b[0] = a.tail.tail;
b[0].tail = new IntList(66, null);
b[2] = b[0].tail;
```

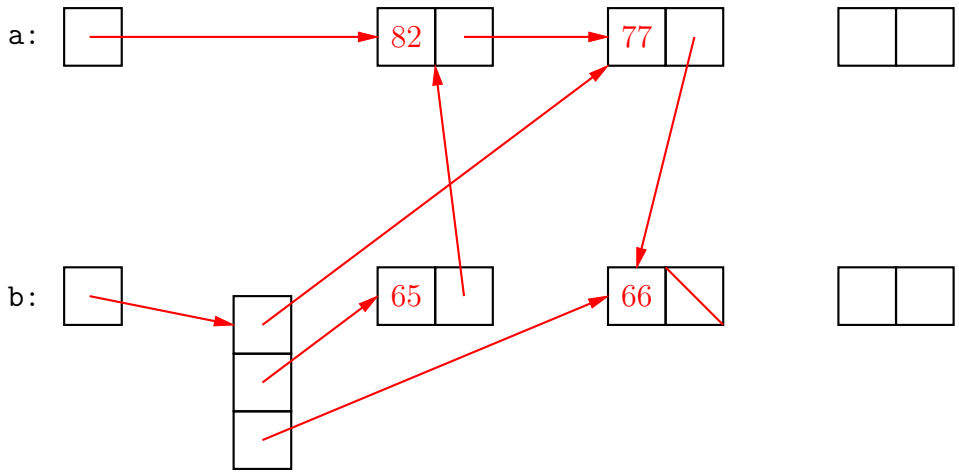
Write Output Here:

```
System.out.println(b[2].head);
```

66

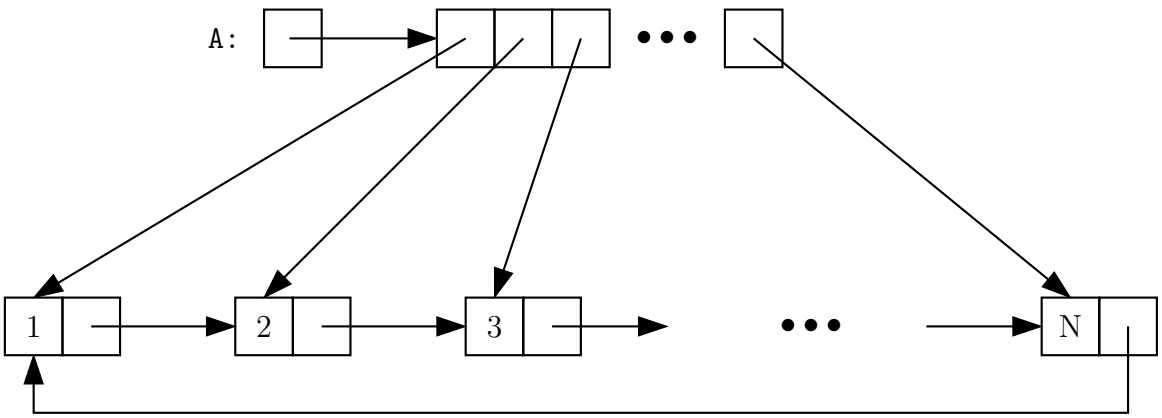
```
System.out.println((b[1].head + 4));
```

69



(c) For this one, fill in the program to match the diagram at the bottom of the page. This time, it is the horizontal group of boxes that denotes an array object. Not all blank lines need to be used.

```
int N = some positive integer (N ≥ 1);
IntList[] A = new IntList[N];

A[N - 1] = new IntList(N, null);

for (int i = N - 2; i >= 0; i -= 1) {

    A[i] = new IntList(i + 1, A[i + 1]);

}

A[N - 1].tail = A[0];
```



2. [3 points] Fill in the method on the next page so that it splits a given 2D array of `ints` into left and right 2D arrays non-destructively about the first entry that equals `m`. There will always be at least one occurrence of `m` in each row. The method returns its result as an array of two 2D arrays. (You might find something helpful on page 2.)

Example:

```
int[][] A = new int[3][4] {
  { 2, 87, -1, 8 },      /* A[0] */
  { 4, 10, 13, -1 },
  { 43, -1, 234, -1 }    /* A[2] */
};

int[][][] R = split(A, -1)
/* R[0] contains                R[1] contains
 *    { { 2, 87 },                 { { 8 },
 *      { 4, 10, 13 },               { },
 *      { 43 } }                     { 234, -1 } }
 */
```

```
/** Return an array [ left, right ], where left and right are 2D
 *  arrays derived from splitting the rows of A around M as described
 *  in the text. There will always be at least one occurrence of M
 *  in each row.
 */
public static int[][][] split(int[][] A, int m) {
    int h = A.length;

    int[][] left = new int[h][]                          ;


    int[][] right = new int[h][]                         ;

    for (int row = 0; row < h; row += 1) {
        int splitCol;


        splitcol = 0                          ;

        while (A[row][splitcol] != m                ) {


            splitcol += 1                     ;
        }


        left[row] = new int[splitcol]                    ;

        right[row] = new int[A[row].length-splitcol-1]   ;

        arraycopy(A[row], 0, left[row], 0, splitcol);

        arraycopy(A[row], splitcol+1, right[row], 0, right[row].length);


        ____________________________________________
    }

    return new int[][][] {left, right};
}
```

3. [2 points] Correct four lines of mistakes in the code on this and the next page so that the `getWeakestMemeTest` unit test on page 9 passes. If a line of code contains a mistake, correct it on the blank below it. Do not correct lines that are not followed by a blank. Leave a blank empty if the preceding line of code is correct.

```
public class Meme {
    String meme;
    int dankness;

    /** DANKNESS must be non-negative. */
    public Meme(String name, int dankness) {
        meme = name;

        _____
        dankness = dankness;

        this.dankness = dankness;
    }

    /** Return the lowest dankness value in MEMES, or 0 if MEMES is null. */
    public static int getWeakestMeme(Meme[] memes) {
        if (memes == null || memes.length == 0) {
            return 0;
        }

        int minDank = 0;

        int minDank = memes[0].dankness;
        for (int i = 0; i <= memes.length; i += 1) {

        for (int i = 0; i < memes.length; i += 1) {
            minDank = Math.min(minDank, memes[i].dankness);

            _____
        }
        return minDank;
    }
```

```
    @Test
    public void getWeakestMemeTest() {
        Meme[] memeStash = new Meme[3];


        ____________________________________
        memeStash[0] = new Meme("Pepe", 4);


        ____________________________________
        memeStash[1] = new Meme("John Cena", 20);


        memeStash[1] = new Meme("John Cena", 120); // Possibility 1
        memeStash[2] = new Meme("Harambe", 20160528);


        ____________________________________
        assertEquals(4, getWeakestMeme(memeStash));

        Meme[] anotherOne = memeStash;


        Meme[] anotherOne = new Meme[1]; // Possibility 2
        anotherOne[0] = new Meme("Only Meme", 100);


        ____________________________________
        assertEquals(100, getWeakestMeme(anotherOne));
    }

    /* Main method omitted */
}
```

4. [2 points] The `BinaryPredicate` interface defines a type of object whose method takes two **int** values and returns a boolean value.

```
public interface BinaryPredicate {
    boolean test(int a, int b);
}
```

We say that two values, x and y, *satisfy* a `BinaryPredicate` if its `test` method returns true when called with the values x and y.

(a) Fill in the definition of the method `allPairsCheck` to fulfill its comment.

```
public class Utils {
    /** If L contains the values [c0, c1, ..., cn], returns true iff
     *  every adjacent pair of values in L (c0, c1), (c1, c2), ...
     *  satifies P.  Always true for lists of length 0 or 1. */
    public static boolean allPairsCheck(BinaryPredicate p, IntList L) {

        for (IntList x = L; x.tail.tail != null; x = x.tail ) {

            if (!p.test(x.head, x.tail.head)) {

                return false;

            }

        return true;
    }
    ...
```

(b) Using `allPairsCheck`, implement the following method (also in class `Utils`).

```
public class Utils { // Continued

    ...

    /** Return true iff the members of L are in non-descending order. */
    public static boolean ascending(IntList L) {

        return allPairsCheck(new Nondescend(), L)              ;
    }

    /* Define any addition methods or classes you need here. */

    class Nondescend implements BinaryPredicate {
        public boolean test(int x, int y) {
            return x <= y;
        }
    }

}
```

5. [2 points] Fill in the blanks as indicated.

```
public class test01 {

    public static void multiplyBy3(int[] A){
        for(int x: A){
            x = x * 3;
        }
    }

    public static void multiplyBy5(int[] A){
        int[] temp = new int[A.length];
        System.arraycopy(A, 0, temp, 0, A.length);

        for(int i = 0; i < temp.length; i += 1){
            temp[i] *= 5;
        }
        A = temp;
    }

    public static void multiplyBy2(int[] A) {
        int[] B = A;
        for (int i = 0; i < B.length; i += 1) {
            B[i] *= 2;
        }
    }

    public static void swap(int A, int B){
        int temp = B;
        B = A;
        A = temp;
    }
```

```
    public static void main(String[] args){
        int[] arr;

        arr = new int[]{2, 3, 3, 4};
        multiplyBy3(arr);

        /* Value of arr: {2, 3, 3, 4                                    } */

        arr = new int[]{2, 3, 3, 4};
        multiplyBy5(arr);

        /* Value of arr: {2, 3, 3, 4                                    } */

        arr = new int[]{2, 3, 3, 4};
        multiplyBy2(arr);

        /* Value of arr: {4, 6, 6, 8                                    } */

        int a = 6;
        int b = 7;
        swap(a, b);

        /* Value of a: 6      Value of b: 7      */
    }

}
```

6. [2 points] In the following classes, cross out the lines that will result in an error (either during compilation or execution). Next to each crossed-out line write a replacement for the line that correctly carries out the evident intent of the erroneous line. Each replacement must be a single statement. Change as few lines as possible.

After your corrections, what is printed from running `java P2.C5`?

Write Output Here:

```
package P1;
public class C1 {

    private int a;
    int b;
    public static int c() {

        return 11;
    }
    public void setA(int v) { a = v; }
    public void setB(int v) { b = v; }
    public int getA() { return a; }
    public int getB() { return b; }

    @Override
    public String toString() {
        return a + " " + getB() + " " + c();
    }
}
```

10 0 11,13 28 11

0 0 11

```
package P1;
public class C2 extends C1 {
    public C2() { }

    public C2(int a, int b) {
        this.a = a;  setA(a);
        this.b = b;
    }
    public static int c() {
        return 12;
    }

    public C1 gen() {
        return new C3();
    }
}
```

```
package P1;
class C3 extends C1 {
    private int a = 10;
    public String toString() {
        return a + " " + getB() + " " + c();
    }

}
-----------------------------------------------
package P2;
public class C4 extends P1.C2 {

    public int getB() {
        return 2 * b;  return 2 * getB();
    }

    public C4(int a, int b) {
        this.a = a;  setA(a);
        this.b = b;  setB(b);
    }

    public C4(int v) {
        this.a = this.b = v;  super(v, v);
    }

}
-------------------------------------------------
package P2;
public class C5 {
    public static void main(String... args) {
        P1.C1 y = new C1();  P1.C1 y = new P1.C1();
        P1.C2 x = new C4(13, 14);
        P1.C3 q = x.gen();  P1.C1 q = x.gen();
        System.out.println(q + ", " + x);
        System.out.println((P1.C2) y);  System.out.println(y);
    }
}
```

7. [1 point] If the earth were a 100-dimensional hypersphere with radius 4000 miles rather than a three-dimensional sphere of that radius, what fraction of its volume would be within 40 miles of its surface?

Since the volume of an n-sphere varies as r^n, the volume of the hyper-earth at depths greater than 40 miles must be $0.99^{100} \approx 0.37$ of the total volume, so that the top 40 miles contains 63% of the total hypervolume.

8. [3 points] Fill in the following `swapHalves` method so that it destructively swaps the first and last halves of an `IntList`. If the list has an odd number of elements, the second part is one longer than the first. Examples:

```
swapHalves(IntList.list(1, 2, 3, 4)).equals(IntList.list(3, 4, 1, 2))
swapHalves(IntList.list(1, 2, 3, 4, 5)).equals(IntList.list(3, 4, 5, 1, 2))
```

Your method must not create new `IntList` objects and must not modify the `.head` fields of any objects. You may not need all lines.

Question unintentionally tricky.

```
/** Destructively swap the first and last halves of the list headed by
 *  L and return the resulting list.  When a list has an odd length,
 *  its first part is taken to be one shorter than its last. */
IntList swapHalves(IntList L) {

    if (L == null || L.tail == null) {

        return L;
    }

    int N = L.size();
    IntList p, result;

    p = L;

    for (int k = 0; k < N/2 - 1; k += 1) {

        p = p.tail;


    }

    result = p.tail;

    for (p.tail = null, p = result; p.tail != null; p = p.tail) {


    }

    p.tail = L;

    return result;
```

```
}
```