Instructions: Use the space provided for each answer. If necessary, use the back of the page. You must complete this exam independently, with no outside resources of any kind. This exam consists of 6 questions, worth a total of 100 points.

Below, we provide the Iterator and (partial) List interfaces, should you need them for reference:

interface Iterator<E>:

// Returns true if the iteration has more elements.
boolean hasNext();

// Returns the next element in the iteration.
E next();

// Removes from the underlying collection the last element returned by the iterator.
void remove();

interface List<E>:

boolean add(E e); // add to the end of the list
void add(int index, E element); // insert at specified index
void clear(); // remove all elements
boolean contains(Object o); // true if list contains the object
E get(int index); // get element at specified index
int indexOf(Object o); // index of first occurrence (or -1)
boolean isEmpty(); // list is empty (size() == 0)
Iterator<E> iterator(); // return iterator though this list
int lastIndexOf(Object o); // index of last occurrence (or -1)
E remove(int index); // remove at specified index
boolean remove(Object o); // remove first occurrence of object
E set (int index, E element); // set item at index to be given element
int size(); // number of elements in list
1. (10 points)
Suppose the following class has been defined:

```java
public class IntRef {
    public int data;

    public IntRef(int data) {
        this.data = data;
    }
}
```

What does the following code print out when run?

```java
IntRef var1 = new IntRef(1);
IntRef var2 = new IntRef(2);
IntRef var3 = new IntRef(3);
var2 = var1;
var2.data = 4;
var1 = var3;
var2.data = 5;
var1.data = 6;

System.out.println("1: " + var1.data);
System.out.println("2: " + var2.data);
System.out.println("3: " + var3.data);
```

**ANSWER:**

1: 6
2: 5
3: 6

(Verified by running the code.)
2. (10 points) What does the following code print out for each input?

```java
public static void mystery(int[] a)
{
    ArrayList<Integer> list = new ArrayList<Integer>();
    int last = 0;
    for (int i = 0; i < a.length - 1; i++) {
        list.add(i/2, last);
        last = a[i];
    }
    System.out.println(list);
}
```

**ANSWERS:**

{1} ___[]_____________________________________________

{1, 2} ___[0]____________________________________________

{1, 2, 3, 4} ___[1, 2, 0]___________________________________

{1, 2, 3, 4, 5, 6, 7, 8} ___[1, 3, 5, 6, 4, 2, 0]______________

(Verified by running the code.)
public class Do {
    public void m1() {
        System.out.println("A1");
    }

    public void m2() {
        m1();
        System.out.println("A2");
    }
}

class Re extends Do {
    public void m1() {
        System.out.println("B1");
    }

    public void m3() {
        System.out.println("B3");
    }
}

class Mi extends Re {
    public void m1() {
        m3();
        System.out.println("C1");
    }

    public void m2() {
        super.m2();
        System.out.println("C2");
    }
}
Suppose the following variables are defined:

```java
Do do1 = new Do();
Do do2 = new Re();
Do do3 = new Mi();
Object obj1 = new Do();
Object obj2 = new Re();
```

Indicate on each line below the output produced by each statement shown. If the statement produces more than one line of output indicate the line breaks with slashes as in a/b/c to indicate three lines of output with a followed by b followed by c. If the statement causes an error (either at compile time or runtime), write the word “error” to indicate this.

**ANSWERS:**

```java
do1.m2();     __A1 / A2___________________________
do2.m2();     __B1 / A2___________________________
do3.m1();     __B3 / C1___________________________
do3.m2();     __B3 / C1 / A2 / C2____________________
do3.m3();     __error___________________________
((Re) do3).m3(); __B3___________________________
((Re) do3).m1(); __B3 / C1___________________________
obj1.m1();    __error___________________________
((Re) obj1).m2(); __error___________________________
((Do) obj2).m1(); __B1___________________________
```

(Verified by running the code)
(Problem instructions by Reges and Stepp.)
4. (20 points) Consider the following class file, Broken.java (with line numbers for your convenience):

```java
import java.util.*;

public class Broken implements Iterable<Object> {
    static String s1;
    protected String s2;

    public static reverse(ArrayList<String> l) {
        int i = 0;
        foreach (s: l) {
            Object s2 = l.get(l.size() - i);
            l.set(i, (String)s2);
            l.set(l.size() - i, s);
            if (i > l.size() / (int)('c' - 'a'))
                return;
        }
    }

    public Iterator<Object> iterator() {
        return (new ArrayList<Object>).iterator();
    }
}
```

(a) Identify all bugs that prevent this class from compiling (syntax errors).

ANSWERS:
- public static reverse is missing return type (void).
- In foreach (s: l), foreach should be for, AND s should be String s.
- new ArrayList<Object> should be new ArrayList<Object>()

(b) The reverse() method is supposed to reverse the contents of an ArrayList of Strings. Identify and correct all bugs that prevent it from doing so (logic errors).

ANSWERS:
- You need to increment i at the end of the loop: i++ between lines 15 and 16.
- l.size() - i should be l.size() - i - 1 in both places.
- if test needs to be changed, e.g.:
  if (i > l.size() / (int)('c' - 'a'))
      return;
  i++;
  (Many other possibilities also work here.)
5. (20 points) Add the method `removeEvens` to `LinkedIntList`, which removes all even numbers from the list. You may not use an iterator. You may declare `ListNode` variables, but you may not construct any new `ListNode` objects. Your method must run in time $O(n)$, where $n$ is the length of the list.

Recall the definitions of the `ListNode` and `LinkedIntList` classes:

```java
public class ListNode {
    public int data;
    public ListNode next;
}

public class LinkedIntList {
    private ListNode front;

    ...
}
```

Here is the declaration of the function you are to implement, as a member of the `LinkedIntList` class:

```java
/**
 * Removes all even numbers from this list.
 */
public void removeEvens()

ANSWERS:

{
    while (front != null && front.data % 2 == 0) {
        front = front.next;
    }
    ListNode curr = front;
    while (curr != null && curr.next != null) {
        if (curr.next.data % 2 == 0) {
            curr.next = curr.next.next;
        } else {
            curr = curr.next;
        }
    }
}
6. (20 points) Write the method `containsInOrder`, which returns true if the first list contains all elements of the second list, *in the same order*. To obtain full credit, your method must run in $O(n)$ time, where $n$ is the length of the list.

**Example:** the list [-1, 2, 8, 14, -1, 4, 0, 2] contains the elements [2, -1, 4] in order. (Elements have been bolded in the first list to show this.) However, the list [4, 0, 2, -1, 8] does not contain the elements [2, -1, 4] in order.

```java
/**
 * Returns true if the first list contains the contents of the second,
 * in order. Elements need not be contiguous.
 *
 * @param l1 list of integers
 * @param l2 subsequence to search for
 * @return true if l1 contains all elements of l2 in the same order
 */
public static boolean containsInOrder(List<Integer> l1, List<Integer> l2)
{
    if (l2.isEmpty()) {
        return true;
    }
    Iterator<Integer> i1 = l1.iterator();
    Iterator<Integer> i2 = l1.iterator();
    int curr2 = i2.next();
    while (i1.hasNext()) {
        int curr1 = i1.next();
        if (curr1 == curr2) {
            if (i2.hasNext()) {
                curr = i2.next();
            } else {
                return true;
            }
        }
    }
    return false;
}
```

ANSWERS:

```java
public static boolean containsInOrder(List<Integer> l1, List<Integer> l2)
{
    if (l2.isEmpty()) {
        return true;
    }
    Iterator<Integer> i1 = l1.iterator();
    Iterator<Integer> i2 = l1.iterator();
    int curr2 = i2.next();
    while (i1.hasNext()) {
        int curr1 = i1.next();
        if (curr1 == curr2) {
            if (i2.hasNext()) {
                curr = i2.next();
            } else {
                return true;
            }
        }
    }
    return false;
}
```