LAST NAME_____________________________first name___________________________ Cornell Net id____________

CS1110 Prelim 2 14 April 2011

This 90-minute exam has 5 questions (numbered 0..4) worth a total of 100 points. Scan the whole test before starting. Budget your time wisely. Use the back of these pages if you need more space. You may separate the pages; we have a stapler at the front of the room.

Question 0 (2 pts). Write your last name, first name, and Cornell NetId, legibly, at the top of each page.

Question 1 (21 pts) Recursion-like. Below is a partial definition of a class `BoolExp`. An instance of `BoolExp` represents a boolean expression. For example, object `a1` below represents the expression

```
true || (false).
```

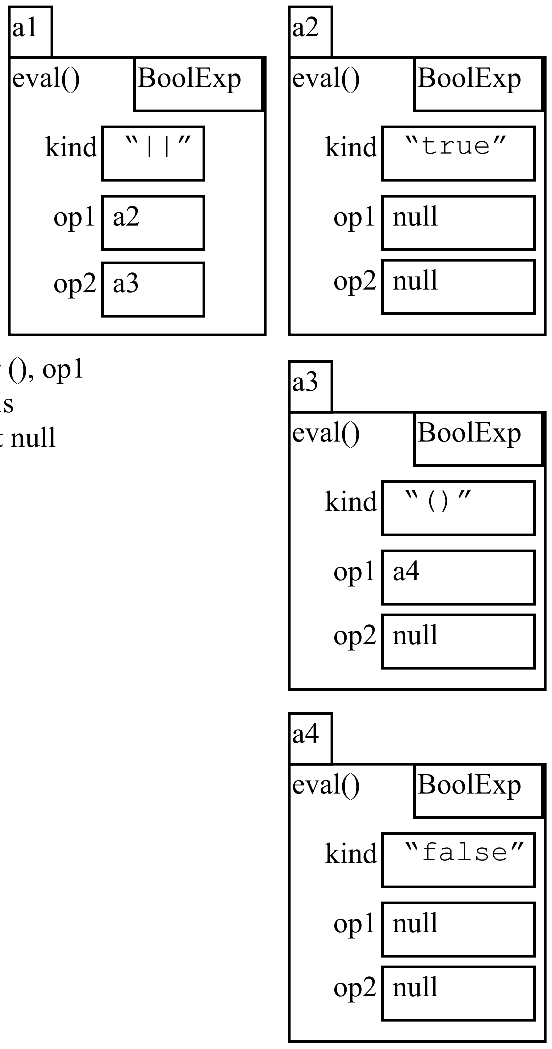
The only method we show is function `eval`, whose purpose is to evaluate the boolean expression and return its value. Write the body of `eval`.

```
/** An instance represents a boolean expression,
    with no negation (!) and no variables. */
public class BoolExp {
  /** kind is one of the strings give below. With
       each, we say what expression this object is:
        "true"    -- the expression is:  true
        "false"   -- the expression is: false
        "&&"      -- the expression is:  op1 && op2
        "||"      -- the expression is:  op1 || op2
        "()"      -- the expression is: ( op1 ) */
   private String kind;  // if kind is &&, ||, or (), op1
   private BoolExp op1;  // is not null. If kind is
   private BoolExp op2;  // && or ||, op2 is not null

   /** = the value of this expression. */
   public boolean eval() {




}
```



LAST NAME_____________________________first name___________________________ Cornell Net id_____________

Question 2 (21 points) for-loops. Write a loop with initialization that stores the odd values of array segment b[m..n] in the beginning of the segment. For example, change the array segment

$$\begin{array}{|ccccccccc|} \multicolumn{1}{c}{m}&&&&&&&&\multicolumn{1}{c}{n}\\ \hline \mathbf{1}&2&4&\mathbf{3}&4&\mathbf{1}&6&8&\mathbf{9}\\ \hline \end{array} \quad \text{to} \quad \begin{array}{|ccccccccc|} \multicolumn{1}{c}{m}&&&&i&&&&\multicolumn{1}{c}{n}\\ \hline \mathbf{1}&\mathbf{3}&\mathbf{1}&\mathbf{9}&4&1&6&8&9\\ \hline \end{array}$$

There is no need to swap; just store the odd values in the front. We don't care what b[i..n] is at the end.

Here are the ground rules. Below, we give a statement that says what to do: the Task. You must

1. Write a postcondition that indicates that the task has been done.
2. Write down here the range of integers (indices) to process: ______________________
3. Write the for-loop with all parts filled in except the repetend.
4. Above the for-loop, write the loop invariant, based on the postcondition.
5. Write initialization (if any).
6. Write the repetend.

Task: Change b[m..n] and store a value in i so that:

b[m..i-1] contains all the odd values in the original array b[m..n]

initialization:

invariant:

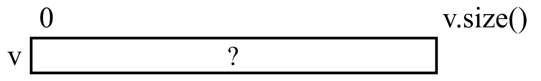
for ( ) {

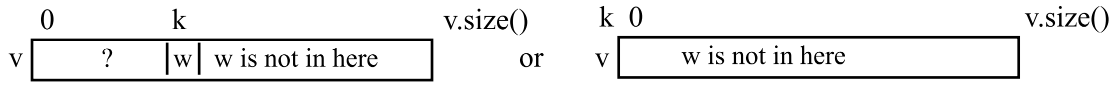
}

postcondition:

LAST NAME_____________________________first name___________________________ Cornell Net id_____________

Question 3 (21 points) while-loops. Implement the essence of `Vector` function `lastIndexOf`: Assume `Vector v` and `Object w` are already initialized and that `k` is already declared but not initialized. Given the precondition below, store a value in `k` so that the postcondition (also below) is true.

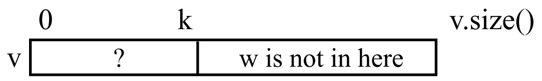
precondition:


postcondition:


Here are the ground rules. Don't write a whole method. Just write *one* while-loop with initialization. The while-loop must use the invariant given below. Do *not* use a return statement anywhere.

Use function `v.get(...)` to get the value v[...].

Function `lastIndexOf` does not test for equality using `v[...] == w`. It uses function `w.equals`. Further, if `w` is **`null`** and some `v[i]` is **`null`**, then v does indeed contain `w`.

invariant:


LAST NAME____________________________first name__________________________ Cornell Net id____________

Question 4 (35 pts) Methods and OO. At the bottom of page 5 are definitions of three classes: `CelestialBody`, `Planet`, and `Star`.

(a) Below, draw the variables declared in the following sequence of four assignments. Then execute the sequence. Draw any objects that are created —do not draw the partition for class `Object`. No room below? Use the back of the previous page or the next page. You may draw Vectors in any reasonable way.

CelestialBody one= **new** CelestialBody(false, "Moon"); **DRAW VARIABLES HERE**

Star two= new Star(**false**, "Sun");

Planet three= **new** Planet(**true**, "Earth");

CelestialBody four= three;

(b) Execute the following statements —changing things as required in the objects you drew.

two.addBody(three);
three.addMoon(one);
two.addBody(four);

(c) To the right of each expression below, write its value:

(1) three **instanceof** CelestialBody

(2) four **instanceof** Star

(3) three == four

(4) one.equals(four)

(5) three.equals(one)

(d) On the back of the previous page, state the two uses of a wrapper class.

CONTINUED ON NEXT PAGE

LAST NAME_____________________________first name___________________________ Cornell Net id____________

Question 4, continued

0	___________	out of 02
1	___________	out of 21
2	___________	out of 21
3	___________	out of 21
4	___________	out of 35
Total	________	out of 100

(e) Below, implement the body of `Planet.equals`. If you have to write a loop, you need not write a loop invariant.

```
/** = "b is a Planet and has the same name, life property, and
      moons as this Planet" */
public boolean equals(Object b) {








}
```

```
/** An instance maintains info about a celestial body */
public class CelestialBody {
   private String name; // Name of the body
   private boolean life; // True if life exists here

   /** Constructor: A Celestial Body with life l, name n*/
   public CelestialBody(boolean l, String n) { ... }

   /** = "this body has life" */
   public boolean hasLife() { ... }

   /** = the name of the body */
   public String getName() { ... }

   /** = “b is a CelestialBody and has the same name
          and life property as this” */
   public boolean equals(Object b) { ... }
}
```

```
/** An instance maintains info about a planet */
public class Planet extends CelestialBody {
   private Vector<CelestialBody> ms; // The moons
                 // of the planet, in alphabetical order

   /** Constructor: A Planet with life l, name n,
        and no moons. */
   public Planet(boolean l, String n) { ... }

   /** Add m to this planet's list of moons */
   public void addMoon(CelestialBody m) { ... }

   /** = the moons of this planet */
   public Vector<CelestialBody> getMoons() { ...  }

   /** = "b is a Planet with the same name, life
          property, and moons as this Planet */
   public boolean equals(Object b) { ... }
}
```

```
/** An instance maintains info about a star */
public class Star extends CelestialBody {
    private Vector<CelestialBody> bs;
         // The bodies that revolve around the star,
         // in alphabetical order

    /** Constructor: Star with life l, name n, and no
         bodies revolving around it. */
    public Star(boolean l, String n) { ... }

    /** Add b to the star's list of orbiting bodies */
    public void addBody(CelestialBody b) { ... }

    /** = a vector of planets orbiting this star */
    public Vector<CelestialBody> getBodies() { ... }
}
```