CSE 331 Winter 2016 Midterm Solution

Name ____

There are 7 questions worth a total of 100 points. Please budget your time so that you get to all of the questions. Keep your answers concise.

The exam is closed book, closed notes, closed electronics, closed classmates, open mind.

Many of the questions have short answers, even if the prompt is a little long. Don't worry!

For all questions involving proofs, assertions, invariants, etc., please assume that all integer quantities are unbounded (e.g., overflow cannot happen) and that integer division is truncating division as in Java, i.e., 5/3 evaluates to 1.

If you do not remember the syntax of some command or the format of a command's output, make the best attempt you can. We will not be grading syntactic details.

Relax and have fun! We're all here to learn.

Please wait to turn the page until everyone is told to begin.

Score ________________ / 100

1. __________________	5. __________________
2. __________________	6. __________________
3. __________________	7. __________________
4. __________________	

Question 1. (5 points) [Forward Reasoning] Using forward reasoning, please write an assertion in each blank space indicating what is known about the program state at that point, given the precondition and previously executed statements. Please simplify your final answers. Be as specific as possible, but be sure to retain all relevant information.

(a)
```
{ x % 2 = 0 && y = x * x }

x = x - 3;

{y = x_pre * x_pre /\ x % 2 != 0 /\ x = x_pre - 3 }

y = y / x;

{y = x_pre^2 / (x_pre - 3) /\ ... }

z = (y + y) * x;

{z = 2 * x_pre^2 /\ ...}
```

(b)
```
{ x % 2 = 0 }

if (x / 2 < 0)

    { x % 2 = 0 /\ x / 2 < 0}

    y = x * x;

    { y = x^2 /\ y % 2 = 0 /\ ... }
else
    { x % 2 = 0 /\ x / 2 >= 0 }

    z = (y + y) * x;

    { z = 2 * y * x /\ ... }

{ ((y = x^2 /\ x/2 < 0) \/ (z = 2 * y * x /\ x / 2 >= 0))
/\
   x % 2 = 0 }
```

Question 2. (5 points) [Weakest Precondition] Using backward reasoning, please find the weakest precondition for each sequence of statements and postconditions below. Insert appropriate assertions in each blank line. Please simplify your final answers.

(a)

```
{ 42^2 < 4 * y * z }

x = 42;

{ x^2 - 4 * y * z < 0 ~~> x^2 < 4 * y *z }

z = x * x - 4 * y * z;

{ z < 0 }
```

(b)

```
{ x > 94 \/ x < -94 \/ x > 50 ~~> x > 50 \/ x < -94 }

if (x / 2 < 0)

    { x > 94 \/ x < -94 }

    x = abs(x) + 6;

    { x > 100 }

else

    { 2x > 100 ~~> x > 50 }

    x = x * 2;

    { x > 100 }

{ x > 100 }
```

Question 3. (10 points) [Specification] Below we have a method which takes 3 integer arrays as input and whose `@ensures` has been specified. Please complete the JavaDoc specification with the preconditions etc. to guarantee the `@ensures`.

```
/**
* @param a : int array
*
*
* @param b : int array
*
*
* @param c : int array
*
*
* @requires : a != null, sorted(a), b != null, sorted(b),
*            c.length = a.length + b.length
*
*
* @modifies : c
*
*
*
* @ensures c contains exactly the sorted contents of a and b
*/
static void merge(int[] a, int[] b, int[] c) {
     int x = 0;
     int y = 0;
     int i = 0;
     while (i < c.length)
          if (x >= a.length)
               c[i] = b[y++];
          else if (y >= b.length)
               c[i] = a[x++];
          else if (a[x] < b[y])
               c[i] = a[x++];
          else
               c[i] = b[y++];
          i++;
}
```

Question 4. (10 points) [Loop Invariants] Please fill in a loop invariant sufficient to prove the postcondition below. Note that within an assertion "`/\`" corresponds to logical conjunction (and), "`\/`" corresponds to logical disjunction (or), "`-->`" corresponds to logical implication, and "`=`" corresponds to mathematical equality (not assignment). You may assume your preconditions on **`a`**, **`b`**, and **`c`** from the previous question hold.

```
int i = 0;

{ (forall j, forall e,
     ((j < x /\ a[j] = e) \/ (j < y /\ b[j] = e))
      --> exists k, c[k] = e) /\
  (forall j, forall k,
     0 <= j < k < i --> c[j] <= c[k]) /\
  x <= a.length /\ y <= b.length /\
  a.length + b.length = c.length }

     while (i < c.length)
          if (x >= a.length)
               c[i] = b[y++];
          else if (y >= b.length)
               c[i] = a[x++];
          else if (a[x] < b[y])
               c[i] = a[x++];
          else
               c[i] = b[y++];
          i++;

{ (forall j, forall e,
     a[j] = e \/ b[j] = e --> exists k, c[k] = e) /\
  (forall j, forall k,
     0 <= j < k < c.length --> c[j] <= c[k]) }
```

Question 5. (15 points) [Termination] Please prove that the `itoa` method below terminates.

Your answer should (a) define function `M(i)` which *always* produces non-negative outputs, (b) show that the output of `M` decreases on every loop iteration (i.e. `M(i)` at the end of the loop is less than `M(i)` at the beginning), (c) show that if `M` produces zero, then the loop terminates (i.e. the loop condition is false when `M(i) == 0`).

```
/**
* @requires i > 0
* @returns the String representation of i
*
* For example: itoa(42) == "42"
*/
public static String itoa(int i) {
     String a = "";
     while (i > 0) {
          String d = Integer.toString(i % 10);
          a = d + a;
          i = i / 10;
     }
     return a;
}
```

(a) Define `M(i)` = abs(i)

(b) Prove `M(i)` decreases on each loop iteration.

If abs(i) > 0, then abs(i / 10) < abs(i).

(c) Prove that if `M(i) == 0`, then the loop condition is false.

if abs(i) == 0, then i <= 0, so the loop exits.

Question 6. (15 points) [Testing] Consider the following method:

```
int foo(int i, int j, boolean x, boolean y) {
     int res = 0;
     if (i < 0) {
          res = j - i;
     } else if (x || y) {
          res = -1;
          if(j == i) {
               res = 1;
          }
     } else {
          res = j;
     }
     return res;
}
```

(a) How many tests do you need to *exhaustively* test this method? Assume there are 2^{32} int values and 2 boolean values.

2^{66}

(b) How many tests do you need to achieve full *statement coverage*?

3

(c) How many tests do you need to achieve full *branch coverage*?

4

(d) In *random testing* we automatically generate random inputs to a method. What is one advantage of random testing? What is one disadvantage?

Random testing may try tests we would not have thought of.
Random testing may waste time on many equivalent / uninteresting tests.

Question 7. (40 points) [ADT] This problem considers an ADT hardened for space. Outside of our protective atmosphere, charged particles blast through the void and can flip bits in computers running on spacecraft. To help mitigate these faults in such a harsh and unforgiving environment, the UW Space League has asked you to design a new implementation of the List interface that stores duplicates of each list element.

When a DupList is constructed, the caller provides the “duplication factor”, the number of each copies of each element that the duplication list should store. When a client adds an element to a DupList, the DupList actually stores n copies of the element. When a client gets the value of list at some index, the DupList checks all the copies of the element at that index and returns the value that occurs most often.

Please specify, implement, and document your version of DupList starting on the following pages.

```
class DupList<E> implements List<E> {

     // DupList fields
     private int factor;
     private List<List<E>> elts;




     /**  TODO: describe your representation invariant (RI)
     *     (Note: for the RI, assume no weird bit flips!)
     *
     *  factor > 0
     *  forall i l, elts.get(i) = l -->
     *     l.length() == factor /\
     *     forall j k, l.get(j) == l.get(k)
     *
     */

     /**  TODO: document and implement the DupList constructor
     *
     *
     *
     *
     *
     *
     *
     *
     */
     public DupList(int factor) {
          this.factor = factor
          this.elts = new ArrayList<List<E>>();
     }
```

```
/**  TODO: document and implement add
*
*
*
*
*/
public boolean add(E elt) {
     List<E> es = new ArrayList<E>();
     for(int i = 0; i < factor; i++)
          es.add(elt);
     elts.add(es);
}

/**  TODO: document and implement get
*    (Note: Remember to return the value of the copy
*           that occurs most often at this index.)
*
*
*
*
*
*/
public E get(int idx) {
     Map<E, Integer> freq = new HashMap<E, Integer>();
     for(E e : elts.get(idx))
          if(freq.containsKey(e))
               freq.put(e, freq.get(e) + 1);
          else
               freq.put(e, 1);
     int maxF = 0;
     E maxE = elts.get(i).get(0);
     for (E k : freq.keySet())
          if(freq.get(k) > maxF) {
               maxF = freq.get(k);
               maxE = k;
          }
     return maxE;
}
```

```
/**  TODO: implement repair
*
*  For each element in the DupList, repair replaces
*  all copies of the element with the value that occurs
*  most often among the copies.
*
*/
public void repair() {
     for(int i = 0; i < elts.size(); i++) {
          E e = get(i);
          for(int j = 0; j < factor; j++) {
               elts.get(i).set(j, e);
          }
     }
}

/**  TODO: implement equals
*    (Note: Two DupLists should be considered equal if they
*           always agree on the value at all valid indices.
*/
public boolean equals(Object o) {
     if(!(o instanceof DupList))
          return false;
     DupList dl = (DupList) o;
     if(factor != dl.factor)
          return false;
     if(elts.size() != dl.elts.size())
          return false;
     for(int i = 0; i < elts.size(); i++)
          if(get(i) != dl.get(i))
               return false;
     return true;
}
```

```
    /**  TODO: Since we overrode equals(), there is another
     *   method we must carefully override. Do so here. */

    public int hashCode() {
         return factor * elts.size();
    }

}
```

(b) Does your implementation of equals() satisfy Java's requirements? Briefly argue why below.

Yes. It is reflexive, symmetric, and transitive.

(c) Suppose we change the specification for the get() method to throw a TooManyBitFlips exception whenever there is there is no majority among the duplicate copies. Is this specification strong, weaker, or incomparable to your specification above? Please explain why.

Incomparable. The behavior makes no additional assumptions about input, no additional guarantees about output, but changes the behavior in some cases by throwing an exception when there is no majority.