CSE 331 Midterm Exam 11/14/16 **Sample Solution**

Remember: For all of the questions involving proofs, assertions, invariants, and so forth, you should assume that all numeric quantities are unbounded integers (i.e., overflow can not happen) and that integer division is truncating division as in Java, i.e., 5/3 => 1.

Question 1. (12 points) (Forward reasoning) Using forward reasoning, write an assertion in each blank space indicating what is known about the program state at that point, given the precondition and the previously executed statements. Your final answers should be simplified. Be as specific as possible, but be sure to retain all relevant information.

(a) (5 points)

```
{ x != 0 }
y = x * x;
{ x != 0 && y = x*x } => y > 0
x = y - 1;
{ x >= 0 && y = x0*x0 && y > 0 }
x = x + y;
{ x > 0 && y = x0*x0 && y > 0 }
```

(Note: the important parts of the postcondition are the final values of x and y.)

(b) (7 points)

```
{ x > 3 && x < 10 }
if (x > y)
   { x>3 && x<10 && x>y }
   y = x - 2;
   { x>3 && x<10 && y>1 && y<8 }
else
   { x>3 && x<10 && x<=y }
   y = y - x;
   { x>3 && x<10 && y>=0 }
{ x>3 && x<10 && ( (y>1 && y<8) || y>=0 ) }
```

(The final postcondition could be simplified further to { x>3 && x < 10 && y>=0 }, but for the test it was fine to leave it as shown above.)

Question 2. (12 points) (assertions) Using backwards reasoning, find the weakest precondition for each sequence of statements and postcondition below. Insert appropriate assertions in each blank line. You should simplify your final answers if possible.

(a) (5 points)

```
{ x-3 >= 1 } => x >= 4
y = x - 3;
{ y*2 >= 1 } => y >= 1 (smallest integer value >= 0.5)
z = y * 2;
{ z+3 >= 4 } => z >= 1
z = z + 3;
{ z >= 4 }
```

(b) (7 points)

```
{ (x!=0 && x!=0) || (x=0 && x!=-1) }
                        => (true || x=0) => true
if (x != 0) {
  { 2*x != 0 } => x != 0
  x = 2 * x;
  { x != 0 }
} else {
  { x+1 != 0 } => x != -1
  x = x + 1;
  { x != 0 }
}
{ x != 0 }
```

The next few questions concern the following partially complete Java class that was begun by one of the summer interns who left before finishing the job. (Not a UW intern, clearly.) The Department of Software Archeology and Reverse Engineering has turned to you for help.

We know that the code is supposed to implement a bag of strings, i.e., a set that allows duplicate elements, sometimes called a multiset. The representation uses an array to hold the elements and has an associated integer variable to record the size of the bag (the number of elements being used in the array). Presumably the array size will be adjusted as needed if (when?) new elements are added to the bag. Here's the code that we have:

```
import java.util.*;

public class StringBag {
  private int size;           // # of strings in this bag
  private String[] items;     // the strings

  // constructor
  public StringBag(String[] vals) {
    size = vals.length;
    items = Arrays.copyOf(vals, size);  // new array copy of vals
  }

  // delete strings with length > n
  public void deleteLongStrings(int n) {
    int k = 0;
    while (k < size) {
      if (items[k].length() > n) {
        items[k] = items[size-1];
        size = size - 1;
      } else {
        k = k + 1;
      }
    }
  }

  // add string to bag (expand bag as needed) and return success
  public boolean add(String s) {
    // not implemented
    return false;
  }
}
```

In the following questions, assume that this code should work as currently written and that it is correct as far as it goes. This code does compile and execute without errors.

Answer questions about this code on the next few pages. You can remove this page from the exam for reference if you'd like.

Question 3. (12 points) (a) (3 points) Give a suitable abstract description of the class as would be written in the JavaDoc comment above the `StringBag` class heading.

A `StringBag` is an unordered collection of `Strings` that may contain duplicates (i.e., a multiset or bag). A typical `StringBag` is {s1, …, sn}, or { } (the empty `StringBag`).

(b) (5 points) Give a suitable Representation Invariant (RI) for this class. (Remember that this RI should be sufficient to guarantee that the existing code executes successfully.)

```
items != null && 0 <= size <= items.length &&
for 0 <= k < size, items[k] != null
```

(It would also be possible to have a RI that allows `items=null && size=0` if the `StringBag` is empty, which is more complicated, but could be done.)

(c) (4 points) Give a suitable Abstraction Function (AF) for this class relating the RI to the abstract value of a `StringBag`.

(Assuming the rep invariant given in (b)):

If `size=0`, this `StringBag` represents the empty bag { }, otherwise the strings in `items[0..size-1]` represent the `StringBag` {`items[0], ..., items[size-1]`}.

(Notes: The `size=0` case above is redundant, but maybe helps the reader. If `size=0`, then `items[0..size-1]` is an empty array section, which means there are no elements in the `StringBag`. If `items=null` is allowed when `size=0` (see part(b), then the AF needs to describe that case as a representation of { }).

Question 4. (12 points) Specification. None of the methods in the `StringBag` are specified properly. Below, supply proper JavaDoc comments for the constructor and `deleteLongStrings` methods for the code on previous pages. Leave any unneeded parts blank. The summary comments at the beginning of each JavaDoc block are supplied for you. Hint: the answers probably won't need all of this space.

```
/** Construct a new StringBag with contents from the
 *  given String array.
 *
 * @param vals array of strings whose elements are the initial
 *             contents of this StringBag
 *
 * @requires vals != null && elements of vals not null(*)
 *
 * @modifies
 *
 * @effects makes a new StringBag with elements vals[0], ...,
 *          vals[vals.length-1].
 *
 * @throws
 *
 * @returns
 *
 */
public StringBag(String[] vals) { constructor implementation omitted }
```

***Note: Other code (`deleteLongStrings`) won't work if any elements are `null`, but this is subtle enough that we didn't take off points if that case was missed. However, `vals` itself cannot be `null`, or else the given constructor code will fail.**

```
/** Delete long strings from this Stringbag.
 *
 * @param n maximum length of strings to be retained in this
 *
 * @requires
 *
 * @modifies this
 *
 * @effects all strings with length > n are removed from this
 *
 * @throws
 *
 * @returns
 *
 */
public void deleteLongStrings(int n) { implementation omitted }
```

Question 5. (16 points) Proof. The implementation of `deleteLongStrings` seems to be okay, but we'd like to be sure. For this problem, give a proof that the code works properly. You will need to provide appropriate assertions, pre- and post-conditions, and loop invariants for your proof. Write your proof in between the lines of code below

```
// delete strings with length > n

public void deleteLongStrings(int n) {

  int k = 0;

  { inv: items[0..k-1] have length <= n and
         items[k..size-1] have unknown length }

  while (k != size) {

    { inv && k!= size }

    if (items[k].length() > n) {

      { inv && items[k].length > n }

      // (note: next two lines effectively delete items[k] from this StringBag)

      items[k] = items[size-1];

      { inv }

      size = size -1;

      { inv }

    } else {

      {inv && items[k].length<=n} => {items[0..k] length <= n}

      k = k + 1;

      { inv }

    }// end if

    { inv }

  } // end loop

  { inv && k=size } => { items[0..size-1] have length <= n }

}
```

Question 6. (12 points, 3 each) Testing. To increase our confidence that the code is correct, we need testing to complement proofs and analysis. For this question, describe four separate, distinct "black box" tests for the `deleteLongStrings` method that you proved correct on the previous page. For each test give the input values and expected result(s). You do not need to write JUnit tests or other Java code – just give a precise, concise description. Also, don't worry about what observer methods might exist – it's good enough to describe the abstract state of the set before and after the test.

There are, of course, many possible tests. Here are four.

(a) **Initialize bag b to { "a", "b", "c" }**
b.deleteLongStrings(3)
Verify that b contains { "a", "b", "c" }

(b) **Initialize bag b to { "xyzzy", "", "ab", "abcd", "abcdefg" }**
b.deleteLongStrings(4)
Verify that b contains {"", "ab", "abcd"}

(c) **Initialize bag b to the empty bag { }**
b.deleteLongStrings(1)
Verify that b is still the empty bag { }

(d) **Initialize bag b to { "abcd", "pqrstuv", "wxyz" }**
b.deleteLongStrings(3)
Verify that b is the empty bag { }

Question 7. (12 points) Something to add. The Software Archeology department is happy with the work you've done so far. But they've discovered another client request that will require an addition to the `StringBag` class. The client would like us to add an observer method that returns an array with the `String`s that are currently in the `StringBag`. We would like to add the following method:

```
// return the current strings in this StringBag to the caller
public String[] getItems() {
  return items;
}
```

(a) (4 points) Is this method correct? In other words, does it return the correct information to the client?

No. It returns the entire contents of the `items` array, even though some or all of the array elements might not be defined since they are in the section of the array `items[size..items.length-1]`, i.e., the part of the array that is not being currently used.

(b) (4 points) Are there any potential representation exposure or other problems with this method? If so, what can go wrong? If not, say so and give a brief reason.

Yes. Client code could alter the contents of the `StringBag` by modifying elements of the returned array, and could cause store nulls in the array, which would violate the representation invariant.

(c) (4 points) If there are problems with this method (identified in parts (a) and/or (b)), describe how to fix them and still provide an observer method that supplies the information desired by the caller. Please describe briefly what needs to be done to fix the problems. You do not need to write any code, but you can if it helps illustrate your answer.

The most reasonable solution is for `getItems` to allocate a new String array with length equal to `size`, copy the contents of `items[0..size-1]` to the new array, and return that new array to the caller. One good way to do this would be to write `return Arrays.copyOf(items,size);`.

(Note: this does not create a representation exposure problem, since `Strings` are immutable. There is no need to make copies of the strings themselves.)

Question 8. (10 points) Comparing specifications. This question does not concern the `StringBag` ADT or any other code from the previous questions.

Here are parts of four possible specifications for a method that has a parameter `n`.

A.
```
@param n
@requires n % 2 = 0 && n > 0
@return an integer > 0
```

B.
```
@param n
@requires n > 0
@return an integer > 0
```

C.
```
@param n
@throws IllegalArgumentException if n % 2 != 0 or n <= 0
@return an integer > 0
```

D.
```
@param n
@requires n > 0
@return an integer >= 0
```

(a) List all of the specification that are stronger than A. __**B, C**__

(b) List all of the specification that are stronger than B. **none**

(c) List all of the specification that are stronger than C. **none**

(d) List all of the specification that are stronger than D. __**B**__

(e) Is it possible for a single method to satisfy A and B? (yes or no) **yes**

(f) Is it possible for a single method to satisfy A and C? (yes or no) **yes**

Question 9. (12 points, 3 each) Overload? Override? You decide! Suppose we have the following class and method definitions.

```
class A {
  void p(int n)    { System.out.println("A.p(int)"); }
  void p(String s) { System.out.println("A.p(String)"); }
  void q()         { System.out.println("A.q()"); }
}

class B extends A {
  void p()                  { System.out.println("B.p()"); }
  void p(int n)   { p("hello"); System.out.println("B.p(int)"); }
  void r(double d)          { System.out.println("B.r(double)"); }
}

class C extends B {
  void p(String s) { System.out.println("C.p(String)"); }
  void q(int n)    { System.out.println("C.q(int)"); }
}
```

For each of the following groups of statements, write down the output produced when the statements are executed or, if there is a compile-time or run-time error, explain in a sentence what is wrong.

(a)
```
C c1 = new C();
c1.p(17);
```

```
C.p(String)
B.p(int)
```

(b)
```
B b1 = new C();
b1.q(17);
```

Error: no `q(int)` method in `B`

(c)
```
B b2 = new B();
A a2 = b2;
a2.p(17);
```

```
A.p(String)
B.p(int)
```

(d)
```
A a3 = new B();
B b3 = a3;
b3.r(3.14);
```

Error: can't assign a3 to b3 without a cast