NAME_____

Exam 1
CSCI 2600 Principles of Software
October 6, 2015

- **DO NOT OPEN THIS EXAM UNTIL TOLD TO DO SO!**
- **READ THROUGH THE ENTIRE EXAM BEFORE STARTING TO WORK.**
- **YOU ARE ALLOWED ONLY 2 "CHEAT" PAGES. NO OTHER MATERIAL IS ALLOWED.**

This exam is worth 150 points.

Make sure you have 12 pages counting this one. There are 3 parts, each including multiple questions for a total of 13 questions. If you need more room for an answer than is provided, please use the back of the page and indicate that you have done so. If you re-do a question, please make clear what is your final answer.

Be clear and brief in your explanations—rambling and lengthy answers will be penalized. All questions have short answers.

The test is longer than those back tests, but you can do it! Relax.

The following is for the use of graders

1. _____/55
2. _____/40
3. _____/55

TOTAL:_____/150

Name: _____

Part I. Reasoning About Code

Note: All variables are **integers.**

Question 1. (5 pts) Order the conditions from strongest to weakest

(a) y = x + 1 && x is even
(b) y is odd
(c) y is odd && x is even
(d) y = 11 && x = 10

Answer: _d , _a , _c , _b

(a) result = 10
(b) 5 ≤ result ≤ 10
(c) 0 ≤ result ≤ 11
(d) 7 ≤ result ≤ 10

Answer: _a , _d , _b , _c

Question 2. (10 pts) Let P => Q => R, S => T => U and { Q } **code** { T } be true. Circle the triples that are true.

a) { P } **code** { S }

b) { P } **code** { U }

c) { R } **code** { S }

d) { R } **code** { U }

e) { Q } **code** { S }

Question 3. (5 pts) Compute the weakest precondition using backward reasoning. Fill in all intermediate conditions at the designated places.

```
P: { (x != 0 && x > 0) || (x = 0 && x > -1) } equiv. to x >= 0
if ( x != 0 ) {
   { _x > 0 _____________ }
   z = x;
   { ___ z > 0 ___________ }
}
else {
   { __x + 1 > 0_____________ }
   z = x+1;
   { ____ z > 0 __________ }
}
Q: { z > 0 }
```

Name: _____

Question 4. (10 pts) Willy Wazoo thinks he can simplify the formula for computing the weakest precondition for if-then-else statements.

Willy claims **wp("`if (b) S1 else S2`", Q) = wp(`S1`,Q) ll wp(`S2`,Q)** should work. Fill in the assignments to **`z`** with values of your choice and compute the precondition P according to Willy's formula. Use your choice to explain why Willy is wrong.

```
P: {  _________________  }
     if (x > 0) {

         z = _________

     }
     else {

         z = _________
     }

Q: { z > 0 }
```

Did this in class. Many solutions are possible.

Name: _____

Question 5. (13 pts) Prove that `int gcd(int a, int b)` computes the correct result if it terminates.

```
// requires: a > 0, b ≥ 0
// returns: the greatest common divisor of a and b.
int gcd(int a, int b) {
    int x = a;
    int y = b;
    int t;
    while (y != 0) {
        t = y;
        y = x%y; // x%y is the remainder in integer division
        x = t;
    }
    return x;
}
```

Loop invariant: gcd(x,y) = gcd(a,b)

Base case:
Before loop x = a and y = b, therefore gcd(x,y) = gcd(a,b) trivially holds.

Inductive case:
Assume that gcd(x,y) = gcd(a,b) holds after k-th iteration. We must show that gcd(x', y') = gcd(a,b) after the k+1-st iteration.

We have:
Fact (0): gcd(x,y) = gcd(a,b) due to inductive hypothesis.
Fact (1): y' = x%y due to statement `y = x%y` in loop
Fact (2): x' = y due to statements `t = y` and `x = t` in loop

(1) => x = q * y + y' where 0 <= y' < y. Therefore, we establish Fact (4): gcd(x,y) = gcd(x,y') = gcd(y,y') due to the basic properties of gcd.
(4) and (2) => gcd(x,y) = gcd(x',y') => gcd(x',y') = gcd(a,b).

Partial correctness:
!(y != 0) and gcd(x,y) = gcd(a,b) equivalent to y = 0 and gcd(x,y) = gcd(a,b) => gcd(x,0) = gcd(a,b). Since gcd(x,0) = x, it follows that x = gcd(a,b).

Name: _____

Question 6. (12 pts) Now prove that `int gcd(int a, int b)` terminates. Code is the same as in Question 5.

```
// requires: a > 0, b ≥ 0
// returns: the greatest common divisor of a and b.
int gcd(int a, int b) {
    int x = a;
    int y = b;
    int t;
    while (y != 0) {
        t = y;
        y = x%y; // x%y is the remainder in integer division
        x = t;
    }
    return x;
}
```

Decrementing function: y.

(1) y decreases. y' = x%y < y by definition.

(2) y is bounded. y >= 0 initially holds, and is preserved through the loop: 0 <= y' < y.

(3) y = 0 => !(y != 0).

Name: _____

Part II. Specifications

Question 7. (5 pts) TRUE/FALSE.

a) (TRUE/**FALSE**) If specification A is stronger than specification B, then any implementation that satisfies B satisfies A as well.

b) (**TRUE**/FALSE) If $P_A \Rightarrow P_B$ and $Q_B \Rightarrow Q_A$ then spec B is stronger than spec A. (P_A denotes the precondition of A and Q_A denotes the postcondition of A.)

Question 8. (15 pts) Below are 4 specifications of function **`double sqrt(double x)`**.

Spec A: requires: $x \geq 0$
returns: y such that $|y*y - x| < 0.001$

Spec B: requires: $x \geq 0$
returns: y such that $|y*y - x| < 0.0001$

Spec C: returns: y such that $|y*y - x| < 0.001$ if $x \geq 0$, and 0.0 if $x < 0$

Spec D: returns: y such that $|y*y - x| < 0.001$ if $x \geq 0$
throws: IllegalArgumentException if $x < 0$

For each of the following pairs of specifications, circle the stronger specification, or circle "Neither" if the two specifications are incompatible or equivalent.

a) A **B** Neither

b) A **C** Neither

c) A **D** Neither

d) B C **Neither**

e) B D **Neither**

f) C D **Neither**

Name: _____

Question 9. (20 pts) Below is the (simplified) Javadoc specification of the **`set`** method from class **`ArrayList`**.

`public E set(int index, E element)`

Replaces the element at the specified position in this list with the specified element.

Parameters:
`index` - index of the element to replace
`element` – element to be stored at the specified position

Returns:
The element previously at the specified position

Throws:
`IndexOutOfBoundsException` - if the index is out of range (`index < 0 || index >= size()`)

a) Convert the Javadoc specification into a PoS specification:

requires: none

modifies: **this[index]**

effects: $this_{post}$[index] = element

returns: $this_{pre}$[index]

throws: IndexOutOfBoundsException if index < 0 ll index >= this.size()

b) Now, convert your PoS specification into a logical formula:

true => (if index < 0 ll index ≥ this.size then
throws IndexOutOfBoundsException
else
$this_{post}$[index] = element and returns $this_{pre}$[index]
)
AND
(foreach i ≠ index, $this_{post}$[i] = $this_{pre}$[i])

Name: _____

Part III. ADTs, Rep invariants, Abstraction functions, Representation exposure

Question 10. (10 pts) TRUE/FALSE

a) (TRUE/FALSE) If every method in a Java class returns only immutable objects then the class is immutable.

b) (TRUE/FALSE) The representation invariant is part of the ADT specification.

c) (TRUE/FALSE) The abstraction function is part of the ADT specification.

d) (TRUE/FALSE) The rep invariant must hold before and after every statement in every method.

e) (TRUE/FALSE) The abstraction function maps valid objects to abstract values.

Question 11. (10 pts) Recall the implementation of **`IntSet.contains`** we saw in class:

```
// requires: none
// modifies: none
// effects: none
// returns: true if x is in this IntSet, false otherwise
public boolean contains(int x) {
    int i = data.indexOf(x);
    if (i == -1)
        return false;
    // move-to-front optimization
    // speeds up repeated membership tests
    Integer y = data.elementAt(0);
    data.set(0,x);
    data.set(i,y);
}
```

The spec of **`ArrayList.contains`** is as follows:

```
// requires: none
// modifies: none
// effects: none
// returns: true if x is in this ArrayList, false otherwise
public boolean contains(Object x)
```

The representation of **`ArrayList`** is as follows:

```
private Object[] elementData; // the array buffer into which the elements of the
                              // ArrayList are stored
private int size; // the size of the ArrayList
```

Is it possible to perform a move-to-front optimization in **`ArrayList.contains`**? Explain.

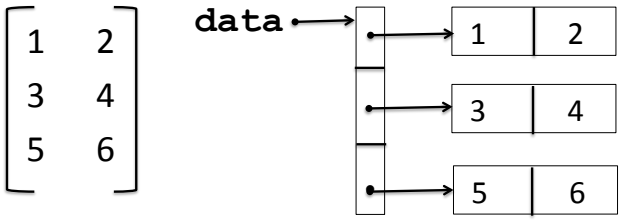
Name: _____

Question 12. (15 pts) Willy Wazoo implemented a matrix with integer elements. (A matrix is a two-dimensional array arranged in rows and columns. An example and a graph elucidating Willy's choice of representation are given on the following page.)

```
public class Matrix {
    private List<List<Integer>> data; // the rep
    private int m; // number of rows
    private int n; // number of columns

    // Rep invariant:

    data != null, m,n >= 0
    data.size() = m;
    data[i].size() = n for each 0 <= i < m
    data[i][j] != null for each 0 <= i < m and 0 <= j < n

    //returns: number of rows in this Matrix
    public int numRows() {
        return m;
    }
    //returns: number of columns in this Matrix
    public int numColums() {
        return n;
    }
    //requires: 0 <= m,n and elts.size() = m*n and elts has no nulls
    public Matrix(List<Integer> elts, int m, int n) {
        data = new ArrayList<List<Integer>>();
        this.m = m;
        this.n = n;

        // Breaks the elts list into m n-element rows
        for (int i=0; i<m*n; i=i+n) {
            // Below, subList returns a view of the portion of elts from index i to i+n-1.
            // Modification to the object referred by elts is visible from row, and vice versa.
            List<Integer> row = elts.subList(i,i+n);
            data.add(row);
        }
    }
    //returns: j-th column in this Matrix
    //throws: IndexOutOfBoundsException if j < 0 || j >= numColumns()
    public List<Integer> getColumn(int j) {
        List<Integer> column = new ArrayList<Integer>();
        // Iterates over the m rows to retrieve the j-th element of each row
        for (int i=0; i<m; i++) {
            column.add(data.get(i).get(j));
        }
        return column;
    }
    //returns: i-th row in this Matrix
    //throws: IndexOutOfBoundsException if i < 0 || i >= numRows()
    public List<Integer> getRow(int i) {
        return data.get(i);
```

```
    }
}
```

A 3-by-2 matrix and an illustration of Willy's representation of it.



a) Write the rep invariant for Willy's implementation at the designated space on the previous page.

See previous page.

b) There is a problem with Willy's implementation. State the problem and suggest a solution.

- Rep exposure in constructor.

Fix: `List<Integer> row = new ArrayList(elts.subList(i,i+n));`

- Rep exposure in getRow.

Fix: `return new ArrayList(data.get(i));`

Name: _____

Question 13. (20 pts) Consider the **IntTree** and **IntList** interfaces specified below.

```
/** An IntTree is a complete binary tree with int nodes with the
 * following property: each node is larger than (or equal to) its
 * children.
 *
 * NOTE: A complete binary tree is a binary tree such that every level
 * except possibly the last is completely filled and all nodes are as far
 * left as possible.
 */



interface IntTree {

  /** Adds val to this tree. Maintains the property. */
  bool insert(int val);
  /** Removes and returns the largest int in this tree.
   *  Maintains the property.
   */
  int remove();
}

/**
 * An IntList is a 1-based array list of ints.
 */
interface IntList {

  /** Adds val at the end of this list. */
  void add(int val);
  /** Replaces the element at specified index with val. */
  void set(int index, int val);
  /** Removes the element at specified index, moves any
   *  subsequent elements to the left.
   */
  void remove(int index);
  /** Returns the element at specified index */
  int get(int index);
  /** Returns the size of this list */
  int size();

}
```

Your task is to implement **IntTree** with an **IntList**. Below is a start:

```
class MyIntTree implements IntTree {
     // Represents the tree
     private IntList rep;

     …
```

Name: _____

Write a rep invariant, abstraction function and sketch the implementation.

a) (5 pts) Rep invariant

The rep invariant enforces the standard heap structure:
rep != null
rep has no nulls
foreach i rep.get(i) >= rep.get(2*i) and rep.get(i) >= rep.get(2*i+1)
(or foreach i rep[i] >= rep[2*i] and rep[i] >= rep[2*i+1])

b) (5 pts) Abstraction function. (You may show your function pictorially.)

The ordered list represents a heap in the standard way:

rep = [a_1, a_2, a_3, … a_n] represents:
a_1 is the root
a_2 to a_3 are the children of a_0, the second "row" in the tree
a_4 to a_7 is the third row,
a_8 to a_15 is the fourth row, etc.

c) (10pts) In 1-2 sentences explain how would you implement **`insert`** and **`remove`**.

Standard implementation of insert and remove in a heap represented as list:

insert: add **val** at last index k, then swap with rep.get[k/2] until either val is at root or val is not larger than its parent rep.get[k/2].

remove: store the element at position 1 (root) in a temp variable. Move val at last index to position 1, then push val down the heap until its place is found. Return temp.