NAME Daniel Schnell

Exam 1
CSCI 2600 Principles of Software
February 26, 2018

- 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 11 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 following is for the use of graders

1. _____/60
2. _____/30
3. _____/60

TOTAL:_____/150

Name: _____

Part I. Reasoning About Code

Note: All variables are **integers.**

Question 1. (6 pts) Order the conditions from strongest to weakest

(a) z == y + 2 && y is odd
(b) z % 2 <> 0 not equal
(c) z is odd && y is odd
(d) z == 11 && y == 9
(e) z <> 0 && y % 1 == 0 not equal

Answer: d, a, c, b, e

(a) x is a non-negative integer
(b) 5 ≤ x ≤ 11
(c) 0 ≤ x ≤ 14
(d) 7 ≤ x ≤ 11
(e) 5 < x ≤ 11

smallest range

Answer: d, e, b, c, a

Question 2. (8 pts) Let P => Q => R, S => T => U and { Q } **code** { T } be true. Which of the following Hoare triples can we assert true? Circle all that apply.

a) { P } **code** { S }	d) { R } **code** { S }	g) { Q } **code** { S }
b) { P } **code** { T }	e) { R } **code** { T }	h) { Q } **code** { T }
c) { P } **code** { U }	f) { R } **code** { U }	i) { Q } **code** { U }

Question 3. (8 pts) Are the following Hoare triples true or false?

a) **TRUE** / **FALSE** { x > 0 && y > 0 } z = x + y { z ≥ 0 }
b) **TRUE** / **FALSE** { x > 0 && y < 0 } z = x + y { z == 0 } not always
c) **TRUE** / **FALSE** { x == 5 && y < -5 } z = x + y { z ≤ 0 } y = -6, Q is true
d) **TRUE** / **FALSE** { x != y } if (y < x) { z = x; x = y; y = z; } { x ≥ y }

T ↙ y < x ↘ F

Name: _____

Question 4. (8 pts) Compute the weakest precondition using backward reasoning. Fill in all intermediate conditions at the designated places.

```
P: { x>1 || x<-1 }
if ( x == 0 ) {
   { rez=0 }
   rez = 0;
   { rez<-1 }
}
else {
   { x>1 }
   if ( x > 0 ) {
      { -x<-1 }
      x = -x
      { x<-1 }
   }
   else {
      { x<-1 }
      // Do nothing
      { x<-1 }
   }
   { x<-1 }
   rez = x*x*x;
   { rez<-1 }
}
Q: { rez < -1 }
```

Name: _____

Question 5. (15 pts) The algorithm below does bitwise addition of `a0` and `b0`. Prove that `int bitwise(int a0, int b0)` computes the correct result if it terminates.

```
// requires: a0 >= 0 && b0 >= 0
// returns: c = a0 + b0
int bitwise(int a0, int b0) {
    int a = a0, b = b0, m = 0, g = 1, c = 0, n;
    while (a > 0 || b > 0) {
        n = m + a%2 + b%2;
        a = a/2;
        b = b/2;
        c = c + (n%2)*g;
        g = g*2;
        m = n/2;
    }
    c = c + m*g;
    return c;
}
```

a	b	m	g	c	n
6	4	0	1	0	/
3	2	0	2	0	0
1	1	0	4	2	1
0	0	1	8	2	2

$c = 2 + 1 \cdot 8 = 10$

Base: $a=0, b=0$, trivially holds

Invariant $(a+b+m) * g + c = a_0 + b_0$

Assume the inv. holds at k^{th} iteration, prove $k+1$ iteration

$(a+b+m) * g + c \Rightarrow (a/2 + b/2 + m)$ where $m = n/2$

$k+1$

$n/2 = \frac{m + a\%2 + b\%2}{2}$ ... dnf

Name: _____

Question 6. (15 pts) Now prove that `int bitwise(int a0, int b0)` terminates. The code is the same as in Question 5.

```
// requires: a0 >= 0 && b0 >= 0
// returns: c = a0 + b0
int bitwise(int a0, int b0) {
    int a = a0, b = b0, m = 0, g = 1, c = 0, n;
    while (a > 0 || b > 0) {
        n = m + a%2 + b%2;
        a = a/2;
        b = b/2;
        c = c + (n%2)*g;
        g = g*2;
        m = n/2;
    }
    c = c + m*g;
    return c;
}
```

Name: _____

Part II. Specifications

Question 7. (8 pts) TRUE/FALSE.

a) (TRUE/FALSE) If specification A is stronger than specification B, then an implementation that satisfies B can be substituted where A is expected.

b) (TRUE/FALSE) Specification B is stronger than A if and only if $P_A \Rightarrow P_B$ and $Q_B \Rightarrow Q_A$. (P_A denotes the precondition of A and Q_A denotes the postcondition of A.)

c) (TRUE/FALSE) Changing a precondition from **`0 < x ≤ 10`** to **`0 <= x < 10`** strengthens the specification. (Assume this is the only change to the specification.)

d) (TRUE/FALSE) Replacing clause requires: **`beginIndex >= 0`** with clause throws: **`IndexOutOfBoundsException if beginIndex < 0`** strengthens the specification.

Question 8. (12 pts) Below is the Javadoc specification of a **`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())`

Convert the Javadoc specification into a PoS specification:
requires: true
modifies: this[index]
effects: this[index] = element
returns: $this_{pre}$[index]
throws: IndexOutOfBoundsException if index<0 || index>=size()
Now, convert your PoS specification into a logical formula:

$$true \Rightarrow (E \wedge (this_{pre}[index] != element))$$

Name: _____

Question 9. (10 pts) Consider 4 specifications of function **`int find(int[] a, int key)`**.

Spec A: requires: **`key`** is in **`a`**
returns: **`index`** such that **`a[index] = key`**

Spec B: returns: **`index`** such that **`a[index] = key`** if **`key`** is in **`a`**,
and **`-1`** if **`key`** is not in **`a`**

Spec C: returns: smallest **`index`** such that **`a[index] = key`** if **`key`** is in **`a`**
throws: **`KeyNotFoundException`** if **`key`** is not in **`a`**

Spec D: requires: **`key`** is in **`a`**
returns: largest **`index`** such that **`a[index] = key`**

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 // no req
b) A (C) Neither // no req
c) A (D) Neither // more out of returns
d) B (C) Neither // has throws, stronger return
e) (B) D Neither // no req
f) (C) D Neither // no req

Name: _____

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

Question 10. (10 pts) TRUE/FALSE

a) (TRUE/FALSE) The rep invariant maps an Object to boolean.
b) (TRUE/FALSE) The rep invariant is part of the ADT implementation.
c) (TRUE/FALSE) The abstraction function is part of the ADT specification.
d) (TRUE/FALSE) The domain of the abstraction function includes some objects that satisfy the rep invariant as well as some objects that do not satisfy the rep invariant.
e) (TRUE/FALSE) The rep invariant must hold before and after every statement in the method.

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.

No because ArrayList needs to be sorted

Name: _____

Question 12. (15 pts) Willy Wazoo is implementing an immutable polynomial with integer coefficients using an IntMap. All specifications are correct; Willy's code, not necessarily so.

```
public class Poly {

    // E.g., { <2,1>,<1,-2>,<0,1> } represents polynomial x^2 – 2x + 1.
    // An empty map represents the ZERO polynomial.
    private Map<Integer,Integer> expToCoeff // rep: exponent-to-coefficient map

    // ADD A SUITABLE REP INVARIANT HERE:
    // Rep inv:
    //

    // ELABORATE THE ABSTRACTION FUNCTION HERE IF NECESSARY:
    // expToCoeffient represents the set of polynomial exponents + coeffients, where an
        empty map represents the zero polynomial
    // requires: input is such that it satisfies rep invariant of Poly
    // effects: creates a new Poly represented by input map
    public Poly(Map<Integer,Integer> input) {
        expToCoeff = input;
    }

    // requires: q != null
    // returns: a new Poly r such that r = this + q
    public Poly add(Poly q) {
        for (Integer key : q.expToCoeff.keySet()) {
            int c1 = 0;
            int c2 = q.expToCoeff.get(key);
            if (expToCoeff.containsKey(key))
                c1 = expToCoeff.get(key);
            expToCoeff.put(key,c1+c2);
        }
        return new Poly(expToCoeff);
    }
}
```

List all that is wrong with Willy's code. One-line bullets is enough. (Since you wrote the rep invariant and abstraction function, not Willy, assume they are correct, and that the code should be written against them.)

- All use of expToCoeffient in Poly add(Poly q) is a representation exposure.
- Should instead use an observer to retrieve the map

Name: _____

Question 13. (25 pts) Consider the **`IntQueue`** and **`IntMap`** interfaces specified below.

```
/** An IntQueue is a queue of integers with capacity C.
* IntQueue can be thought of as an ordered list of integers:
*
* @spec_field queue = [a_1, a_2, … a_n] where n <= C.
*
*/

interface IntQueue {
/** Adds val at the end of this queue if queue is not full:
if queue_pre = [a_1, …, a_n] then queue_post = [a_1 ,…, a_n, val] */
  bool enqueue(int val);

/** Removes and returns the first element in this queue if queue is not
empty: if queue_pre = [a_1, a_2 …, a_n] then queue_post = [a_2 ,…, a_n] */
  int dequeue();
}


/** An IntMap is a mapping from integers to integers.
* IntMap can be thought of as a set of key-value pairs:
*
* @specfield pairs = { <k1, v1>, <k2, v2>, <k3, v3>, ... }
*/

interface IntMap {

/** Associates specified value with specified key in pairs. */
  bool put(int key, int value);

/** Removes the mapping for key from pairs if it is present. */
  void remove(int key);

/** Returns true if pairs contains a mapping for the specified key. */
  bool containsKey(int key);

/** Returns the value to which specified key is mapped, or 0 if this map
contains no mapping for the key. */
  int get(int key);
}
```

Name: _____

Your task is to implement **IntQueue** using an **IntMap** rep. Below is a start. You may add additional representation fields if needed. Your rep fields, rep invariant and abstraction function should give a strong hint about what the implementation is going to be.

```
class MyIntQueue implements IntQueue {

    private int C; // capacity
    private IntMap rep;
    …
```

a) **(8 pts)** Write a suitable rep invariant:

rep.size() ≤ C;

b) **(7 pts)** Write an abstraction function. (You may show your function pictorially.)

Int Queue represents a mapping of a queue of integers to integers

queue [<K1, V1>, <K2, V2>, <K3, V3> ... <Kc, Vc>]

c) **(10pts)** Implement **enqueue** OR **dequeue** and prove that it preserves the rep invariant.

```
bool enqueue(int val){
    if rep.containsKey(val){ return false;}
    // checks size of queue
    int count=0;
    for(int x : rep.keySet()){
        count++;
    }
    if(count < C){
        rep.put(val, val);
        return true;
    }
    else { return false}
```