MIDTERM TEST 1
CSCI 2600 Principles of Software
February 24, 2020

- **DO NOT OPEN THIS TEST UNTIL TOLD TO DO SO!**
- **READ THROUGH THE ENTIRE TEST BEFORE STARTING TO WORK.**
- **YOU ARE ALLOWED ONLY ONE "CHEAT" SHEET WHICH IS STAPLED TO THE BACK OF YOUR TEST. DO NOT SEPARATE YOUR "CHEAT" SHEET FROM THE TEST.**
  NO OTHER MATERIAL IS ALLOWED.

This exam is worth 100 points.

Make sure you have 12 pages, counting this one, plus your "cheat" sheet, if you provided one. There are 2 parts, each including multiple questions for a total of 10 questions. If you need more room for an answer than is provided, please use the blank page 12 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.

Part I. Reasoning About Code

Note: All variables are **integers.**

Question 1. (12 pts) Order the conditions from strongest to weakest

(a) z = y + 3 && y is even
(b) z % 2 <> 0
(c) z = 11 && y = 8
(d) z is odd && y is even
(e) z <> 0 && y % 1 == 0

Answer: _ , _ , _ , _, _

(a) $8 \leq z \leq 14$
(b) $0 \leq z \leq 16$
(c) z is a non-negative integer
(d) $5 \leq z \leq 14$
(e) false

Answer: _ , _ , _ , _, _

(a) x is a day in February, 2020
(b) x is February 24, 2020
(c) x is a day in February
(d) x is a Monday in February, 2020
(e) x is a day in 2020

Answer: _ , _ , _ , _, _

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

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

Question 3. (10 pts) For each of the following Hoare triples, indicate which are valid and which are not valid (not necessarily true). If a triple is not valid, give a counter example. All variables are ints.

a) (VALID / NOT VALID) { x ≥ 0 && y < 0 } z = x * y { z < 0 }

b) (VALID / NOT VALID) { x ≥ 0 && y ≤ 0 } z = x + y { z == 0 }

c) (VALID / NOT VALID) { true } x = z * 3; { x ≥ z }

d) (VALID / NOT VALID) { x != y } if (y < x) { z = x; x = y; y = z; } { y ≥ x }

e) (VALID / NOT VALID)

```
{ w is odd && |w| >= 2 }
if (w <= 0)
    x = w * 2 + 1;
    x = x * x + 1;
else
    x = w + 5;
{ x is even && x != 2}
```

Question 4. (10 pts) Compute the weakest precondition using **backward** reasoning. Fill in all intermediate conditions at the designated places.

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

Question 5. (9 pts) Compute the strongest postcondition using **forward** reasoning. Fill in all intermediate conditions at the designated places. Carry all variables forward. Simplify your final postcondition. Assume all variables are ints. Simplify the postcondition.

a)

```
{ |x| > 5 }
y = 5;
{ ____________________________________________________ }

x = x * 3;
{ ____________________________________________________ }

x = x - 2;
{ ____________________________________________________ }
```

b)

```
{ x > 0 && z > 4 }
if (x < 5 && z < x) {
     { _____________________________________________ }

     y = z + x;
     { _____________________________________________ }

}
{ _____________________________________________ }
```

Question 6 (16 pts)

Our friend Willie Wazoo has created the method shown below. Willie thinks this code works properly. That is, he thinks it calculates a0 + b0 when the preconditions are met, but is not completely sure it works. We will help Willie out by proving that given that the preconditions hold, the postcondition will hold.

```
// precondition a0 >= 0 && b0 >= 0
public static int bitwise(int a0, int b0) {
    int a = a0, b = b0, m = 0, g = 1, c = 0, n = 0;
    System.out.println("a" + "\t" + "b" + "\t" + "n" + "\t" + "m" +
                       "\t" + "a+b+m" + "\t" + "g" + "\t" + "c");
    System.out.println(a + "\t" + b + "\t" + n + "\t" + m + "\t" +
                        (a + b + m) + "\t" + g + "\t" + c);
    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;
        System.out.println(a + "\t" + b + "\t" + n + "\t" + m +
                       "\t" + (a + b + m) + "\t" + g + "\t" + c);
    }
    c = c + m * g;
    return c;
}   // postcondition: c == a0 + b0
```

To help understand the code, Willie tested the code with bitwise(18, 3) and generated this table:

a	b	n	m	a + b + m	g	c
18	3	0	0	21	1	0
9	1	1	0	10	2	1
4	0	2	1	5	4	1
2	0	1	0	2	8	5
1	0	0	0	1	16	5
0	0	1	0	0	32	21

a) (2 points) Identify the loop invariant. Willie's table may give a hint.

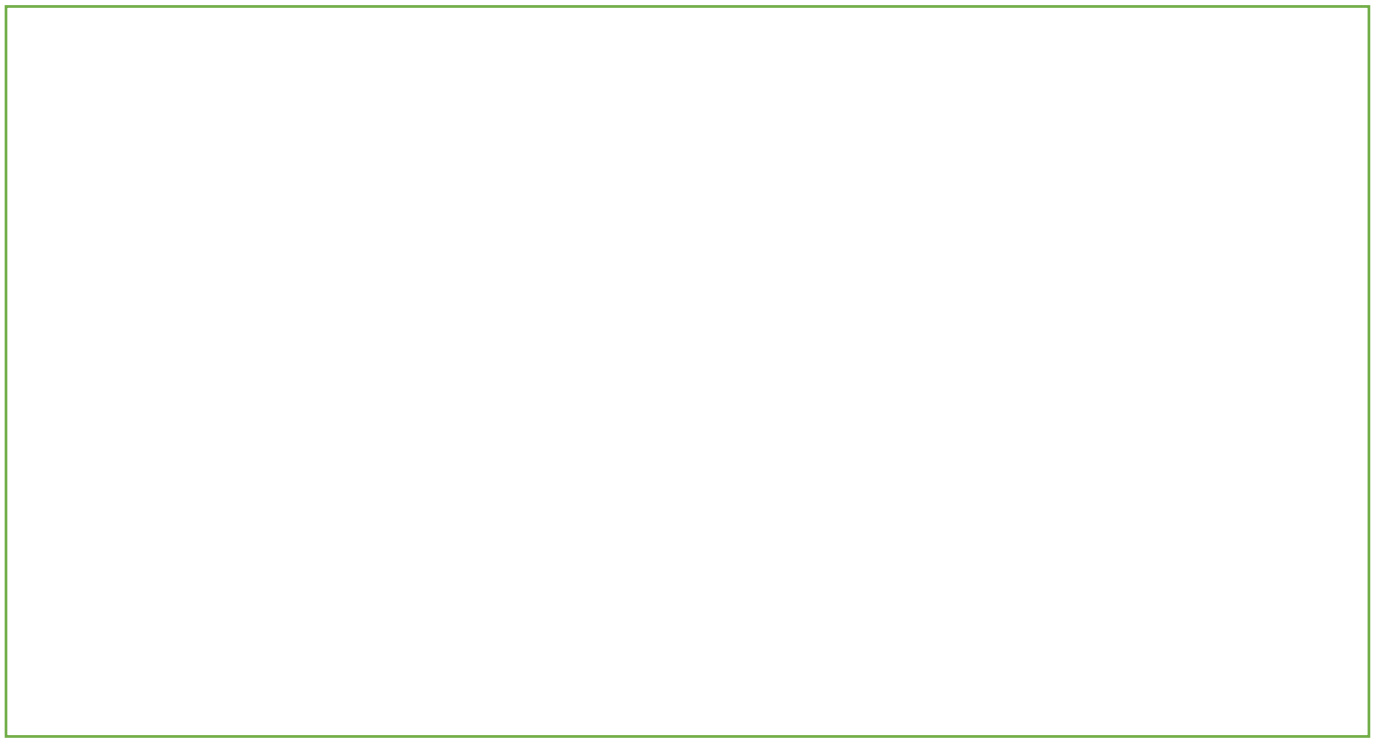
b) (2 points) Using the loop invariant you defined above, show that the loop invariant holds before the first iteration of the loop, i.e., show that the base case holds.

c) (8 points) Assume that the invariant is true at some iteration k and show by **induction** that it is true at the next iteration.

d) (2 points) Show that the negation of the loop condition and the loop invariant imply the postcondition. That is, !(a > 0 || b > 0) && LI => c == a0 + b0.

e) (2 points)

Choose a decrementing function D and show that it is non-negative before the first iteration and decreases with each iteration. Show that when D is at its minimum, the loop exit condition is implied.



Part II. Specifications

Question 7. (7 pts)

a) (TRUE/FALSE) If specification A is stronger than specification B, then any client that meets the obligations of A, meets the obligations of B as well.

b) (TRUE/FALSE) A loop invariant must be true after each statement in a loop.

c) (TRUE/FALSE) You can weaken a specification by strengthening the postcondition, while keeping the precondition the same.

d) (TRUE/FALSE) If specification A consists of precondition P_A and postcondition Q_A and specification B consists of precondition P_B and postcondition Q_B, then P_B => P_A && Q_A => Q_B is a sufficient and necessary (i.e., “if and only if”) condition for A to be stronger than B.

e) (TRUE/FALSE) In the private method, you should always check if the client fails to provide the preconditions.

f) (TRUE/FALSE) In the public method, you should check if the client fails to provide the preconditions unless such a check is expensive.

g) (TRUE/FALSE) In Java, the “equals” method always compares the contents of two objects.

Question 8. (12 pts) Consider 4 specifications of function **`int find(int[] a, int key)`**.

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

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

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

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

b) A C Neither

c) A D Neither

d) B C Neither

e) B D Neither

f) C D Neither

Question 9. (10 pts) Below is the Javadoc specification of the add() method from PriorityQueue<E>.

`public boolean add(E e)`

Inserts the specified element into this priority queue.

Parameters:
`e` - the element to add

Returns:
`true` (as specified by Collection.add(E))

Throws:
`ClassCastException` - if the specified element cannot be compared with elements currently in this priority queue according to the priority queue's ordering
`NullPointerException` - if the specified element is null

Rewrite the specification in Principles of Software style. You don't have to use formal notation, but you can if you wish. A few concise sentences will suffice. You may circle any words or lines ①

from the JavaDoc above, tag them with a number, and then refer to this number in your solution below.

Requires:

Modifies:

Effects:

Returns:

Throws:

Question 10 (6 points)

Given the following method signature and specification, write the specification as a logical formula in the
`R => (E ∧ (nothing but M is modified) )` format.

```
static void uniquefy(List<Integer> lst)

requires: lst is non-null.
modifies: lst
effects: removes duplicate elements from lst, maintains order.
         E.g., <1,1,2,2,2,3,1> becomes <1,2,3>
returns: none
throws: none
```