CSE 331: Software Design & Implementation Spring 2026

Homework 4

Due: May 6th, 11:59pm

Task 1 – Assertificate of Strength [6 pts]

In this task, you will consider the following assertions about an array, `L`, of integers.

```
// Assertion A
L[i] > 0 for any 0 <= i < len(L)

// Assertion B
L[i] > 0 for any 0 <= i < len(L) - 1

// Assertion C
L[i] >= 0 for any 0 <= i < len(L)

// Assertion D
L[i] is even for 0 <= i < len(L) / 2

// Assertion E
L[i] is even for any 0 <= i < len(L) where i is even

// Assertion F
L[i] is even for any 0 <= i < len(L)
```

Fill in the blanks below to state whether the assertions are weaker, stronger, or incomparable.

Note that this question asks about *assertion* strength, not specification strength. Recall that an assertion P is *stronger* than an assertion Q if P implies Q.

Assertion A is _____ than Assertion B.

Assertion A is _____ than Assertion C.

Assertion B is _____ than Assertion C.

Assertion D is _____ than Assertion E.

Assertion D is _____ than Assertion F.

Assertion E is _____ than Assertion F.

Task 2 – Arrayzing Profits [12 pts]

Faraz has decided to expand his startup to a new field: quantitative finance. He has defined a method that takes in an array of stock prices. Each element in the array represents the price of the stock on a certain day. Faraz can choose to buy stock on any day and sell it on that day or any day after. If he chooses to buy on day i and sell on day $j \geqslant i$, his profit is prices$[j]$ − prices$[i]$.

Faraz has defined a method that can determine the maximum achievable profit in $\mathcal{O}(n)$ time. The spec for the method is given below:

```
/**
 * Determines the maximum profit that can be achieved with
 * the given stock prices.
 * @requires prices is not null and nonempty
 * @param prices an array representing the stock prices at each day.
 * @return the maximum possible profit over these stock prices
public static int maxProfit(int[] prices);
```

In his implementation, Faraz uses `Math.max` and `Math.min`. The specs for both methods are below:

```
/**
 * Returns the greater of two int values. That is, the result is the
 * argument closer to the value of Integer.MAX_VALUE. If the arguments
 * have the same value, the result is that same value.
 * @param a an argument.
 * @param b an argument.
 * @return the larger of a and b.
public static int max(int a, int b);

/**
 * Returns the smaller of two int values. That is, the result is the
 * argument closer to the value of Integer.MIN_VALUE. If the arguments
 * have the same value, the result is that same value.
 * @param a an argument.
 * @param b an argument.
 * @return the smaller of a and b.
public static int min(int a, int b);
```

The code for the method is given below. Recall that this algorithm runs in linear time. It may feel more natural to simply compare all possible buy-sell date pairs and simply choose the one with the biggest profit margin. However, this takes quadratic time, so this algorithm is doing something rather nontrivial

to improve efficiency:

```
public static int maxProfit(int[] prices) {
   int maxProfit = 0;
   int lowestPrice = prices[0];
   int i = 1;
  {{ P1 : _____ }}
  {{ Inv: 1 ⩽ i ⩽ prices.length and lowestPrice = min(prices[:i]) and
     maxProfit is the maximum profit over prices[:i] }}
   while (i < prices.length) {
     {{ P2 : _____ }}
     {{ Q2 : _____ }}
      maxProfit = Math.max(maxProfit, prices[i] - lowestPrice);
     {{ _____ }}
      lowestPrice = Math.min(lowestPrice, prices[i]);
     {{ _____ }}
      i++;
     {{ Inv: 1 ⩽ i ⩽ prices.length and lowestPrice = min(prices[:i]) and
        maxProfit is the maximum profit over prices[:i] }}
   }
  {{ P3 : _____ }}
  {{ Q3 : maxProfit is the maximum profit over prices }}
   return maxProfit;
}
```

a) Use forward reasoning to fill in assertion P_1. Then explain in a few sentences why P_1 implies Inv.

b) Fill in P_3 and explain in a few sentences why P_3 implies Q_3.

c) Fill in P_2. Then fill in Q_2 using backward reasoning. Explain in a few sentences why P_2 implies Q_2.

Task 3 – Kelvin and Hobbes [16 pts]

In this task, you will implement the following function, which takes an array containing temperatures in Fahrenheit and converts it into an array of the same temperatures in Kelvin.

If a temperature t is in Fahrenheit, first compute the Celsius value: `5 * (t - 32) / 9`, then add 273 to obtain the temperature in Kelvin.

```
/**
 * Converts each temperature in A from Fahrenheit to Kelvin (in place).
 * @param A an array of Fahrenheit temperatures (must not be null)
 * @modifies A
 * @effects A is an array of the same temperatures as A_0 but in Kelvin.
 */
public void fahrToKelvin(int[] A);
```

With each loop invariant below, fill in the missing parts of the code to make it correct with the **given invariant**. (If the code works correctly with some other invariant, it is not correct.)

a)
```
int i = _________________;

// Inv: A[:i] is converted to Kelvin and A[i:] = A_0[i:]
while (_________________) {

}
```

b)
```
int i = _________________;

// Inv: A[i:] is converted to Kelvin and A[:i] = A_0[:i]
while (_________________) {

}
```

c)
```
int i = _________________;

// Inv: A[:i-1] is converted to Kelvin and A[i-1:] = A_0[i-1:]
while (_________________) {

}
```

d)
```
int i = _________________;

// Inv: A[i+1:] is converted to Kelvin and A[:i+1] = A_0[:i+1]
while (_________________) {

}
```

The next few problems concern the following ADT:

```
/**
 * Represents a *mutable* map with integer keys and values.
 * Think of this as a list of integer pairs (an association list).
 */
public interface MutableIntMap {
  /**
   * Update the map so that k maps to v. The map should contain all
   * existing mappings in obj when the key is not equal to k.
   * @param k the key to add or update
   * @param v the value for key k
   * @modifies obj
   * @effects obj with the value of k replaced with v if k is contained,
   *          or obj with new (k, v) pair if not contained
   */
  public void put(int k, int v);

  /**
   * Returns the corresponding value of key k.
   * @param k the key whose value is to be retrieved
   * @requires obj to contain k
   * @return v the value for key k
   */
  public int get(int k);

  /**
   * Returns whether the map contains a mapping for key k.
   * @param k the key to check
   * @return true if the mapping exists, false otherwise
   */
  public boolean contains(int k);

  /**
   * Removes the mapping for k if such a mapping exists in obj.
   * Otherwise do nothing
   * @param k The key to search for and remove.
   * @modifies obj
   * @effects obj with the mapping for k removed if k is contained
   *          or obj if not contained
   */
  public void remove(int k);

  /**
   * Removes all key value mappings from the map.
   * @modifies obj
```

```
   * @effects obj is empty
   */
  public void clear();

  /**
   * Returns the number of key-value pairs in the map.
   * @return the number of key-value pairs in the map.
   */
  public int size();
}
```

Task 4 – Cyberpunk Testrunners [9 pts]

In this task, you will be testing `MyMap`, an implementation of `MutableIntMap`.

```
public class MyMap implements MutableIntMap {
    private int[] keys;
    private int[] values;
    private int size;

    // RI: keys.length = values.length, size <= keys.length, keys has no duplicates
    // AF: obj is the map of the keys[:size] to their corresponding (same index) values

    /**
     * Creates a new MyMap from two arrays of keys and values and an int size.
     * @param keys the keys of the map
     * @param vals the values of the map
     * @param size the size of the map
     * @requires len(keys) = len(vals)
     *           keys has no duplicate entries
     *           size <= keys.length
     * @effects obj is the map of the given keys[:size] to their corresponding
     *          (same index) values
     */
    public MyMap(int[] keys, int[] values, int size) {
        if (keys.length != values.length) {
            throw new IllegalArgumentException("keys and vals not same length");
        }
        this.keys = Arrays.copyOf(keys, size);
        this.values = Arrays.copyOf(values, size);
        this.size = size;
    }

    ...

    /**
     * Removes the mapping for k if such a mapping exists in obj.
     * Otherwise do nothing
     * @param k The key to search for and remove.
     * @modifies obj
     * @effects obj with the mapping for k removed if k is contained
     *          or obj if not contained
     */
    public void remove(int key) {
        for (int i = 0; i < keys.length; i++) {
            if (this.keys[i] == key) {
                this.keys[i] = this.keys[this.size - 1];
```

```
                this.values[i] = this.values[this.size - 1];
                this.size -= 1;
            }
        }
    }
}
```

Write JUnit tests for the implementation of `remove` which satisfy 331 testing requirements. You may assume the all other methods including the constructor and observers work as specified. Feel free to use or not use the given starter code.
Hint: Note that since `remove` is a mutator method, we must *observe* the side effects on the abstract state to properly test it!

```
    @Test
    public void testRemove() {
        MyMap m = new MyMap(new int[]{1, 2}, new int[]{10, 20}, 2);
    }
```

Task 5 – Map, Crackle, and Pop [12 pts]

Consider another implementation of `MutableIntMap`, `ListPairMap`. This implementation stores the keys and values in two separate lists of the same length.

```
public class ListPairMap implements MutableIntMap {
    // RI: len(keys) = len(vals) and keys has no duplicates
    // AF: obj is the map of keys to their corresponding (same index) values
    private List keys;
    private List vals;
}
```

The `List` type is our standard singly linked list:

```
private static class List {
    public final int hd;
    public final List tl;
    public List (int hd, List tl) {
        this.hd = hd;
        this.tl = tl;
    }
}
```

You may assume `checkRep()` is implemented correctly: it throws an exception if the RI does not hold, and does nothing otherwise.

For each of the following implementations, evaluate:

1. **Correctness**: Does the code satisfy the method's specification and maintain the RI? Briefly explain why.
2. **Safe implementation practices**: Does the code follow 331-style defensive programming? Specifically:
   - `checkRep()` should be called at the beginning and end of every mutator method.
   - `checkRep()` should be called at the beginning of every observer method.
   - Preconditions should be checked defensively when possible.
   - The implementation should not expose its internal representation to clients.

   Briefly explain why or why not.

a) Consider the following implementation of the constructor:

```
/**
 * Creates a new ListPairMap from a list of keys and values.
 * @param keys the keys of the map
 * @param vals the values of the map
 * @requires len(keys) = len(vals) and keys has no duplicate entries
 * @effects obj is the map of the given keys to their corresponding
 *          (same index) values
 */
public ListPairMap(java.util.List<Integer> keys,
                   java.util.List<Integer> vals) {
    if (keys.size() != vals.size()) {
        throw new IllegalArgumentException(
            "arguments must have same length");
    }
    this.keys = null;
    this.vals = null;

    for (int i = keys.size() - 1; i >= 0; i--) {
        this.keys = new List(keys.get(i), this.keys);
        this.vals = new List(vals.get(i), this.vals);
    }
    checkRep();
}
```

b) Consider the following implementation of `get`:

```
public int get(int k) {
    if (!contains(k)) {
        throw new IllegalArgumentException("map must contain k");
    }
    List ks = this.keys;
    List vs = this.vals;
    while (ks != null && ks.hd != k) {
        ks = ks.tl;
        vs = vs.tl;
    }
    return vs.hd;
}
```

c) Consider the following implementation of `put`:

```
public void put(int k, int v) {
    this.keys = new List(k, this.keys);
    this.vals = new List(v, this.vals);
    checkRep();
}
```

d) Consider the following implementation of `clear`:

```
public void clear() {
    List clearedKeys = null;
    List clearedVals = null;
}
```