CSE 331: Software Design & Implementation Fall 2025

Homework 7

Due: Friday, November 14th, 6pm

Task 1 – Another One Bites the Rust [12 pts]

Answer the following questions about mutable and immutable values.

a) In the Rust programming language, which of the following combinations are *not* allowed? Does this match or differ from the general advice given in lecture?

	one reference	**multiple references**
mutable variable		
immutable variable		

b) Consider the following two functions that take a parameter of the given `struct` type:

```
struct Value { number: i32 }

fn f1(value: Value) { ... }

fn f2(value: &mut Value) { ... }
```

The first snippet of code has an error. Why is this an error but the second snippet not?

```
let mut x = Value {number: 10};
f1(x);
let mut y = x;  // error :(
```

```
let mut x = Value {number: 10};
f2(&mut x);
let mut z = x;       // okay
```

c) Consider the following Rust code that works with fractions:

```
/// Represents the fraction numer / denom.
/// Note that, e.g., 4 / 3 and 8 / 6 represent the same number.
struct Fraction {  numer: i32, denom: i32 }

/// Checks for equality of the given fractions.
/// Note that 4*6 = 24 = 8*3, so this confirms 4/3 = 8/6.
fn eq(first: &Fraction, second: &Fraction) -> bool {
    return first.numer * second.denom == second.numer * first.denom;
}
```

Rust will not let me call the following function on a value that is currently being stored as a key in a HashMap (with the map having ownership). Why not?

```
fn normalize(frac: &mut Fraction) {
    let d = gcd(frac.numer, frac.denom);
    frac.numer /= d;
    frac.denom /= d;
}
```

d) Why is Rust's restriction inappropriate in this case?

e) The following Python function finds the indexes in the given list (if any) where the two consecutive numbers are the same.

For example, if given the list $[1, 2, 3, 4, 1, 2, 3, 4]$, it would return $[0, 4]$ because the pair of numbers at index 0 are $[1, 2]$ and pair of numbers at index 4 are $[1, 2]$, which are the same.

```
def repeated_pair(A):
    index = {}
    for i in range(0, len(A)-1):
        nums = [A[i], A[i+1]]
        if nums in index:
            return [index[nums], i]
        index[nums] = i
    return None
```

Unfortunately, when run, this code does not work. Why not? How should it be fixed?

f) The following Java function aims to do the same thing as the Python function above.

```
public static int[] repeatedPair(int[] A) {
    Map<int[], Integer> index = new HashMap<>();

    for (int i = 0; i < A.length - 1; i++) {
        int[] nums = new int[] { A[i], A[i + 1] };
        if (index.containsKey(nums))
            return new int[]{index.get(nums), i};
        index.put(nums, i);
    }
    return null;
}
```

Unfortunately, it does not work either. Why not?

The remaining problems work with the following interface:

```
/**
 * Represents a collection of integers.
 *
 * Clients can think of a set as a list of integers that has **no dups**.
 * The order of the items in the list is not important as contains behaves
 * identically with any ordering.
 */
public interface IntSet {
   /**
     * Determines whether the given number is in the set.
     * @param n The number to look for
     * @return contains(obj, n), where
     *     contains(nil, n)    := false
     *     contains(m :: L, n) := true            if m = n
     *     contains(m :: L, n) := contains(L, n)  if m /= n
     */
    boolean contains(int n);

    /**
     * Returns the size of the set.
     * @return len(obj)
     */
    int size();

    /**
     * Returns the elements of this set in sorted order.
     * @return a **sorted** list L such that
     *     contains(n, L) = contains(n, obj) for any integer n
     */
    int[] elements();
 }
```

The following is a mutable version of the same thing. It extends `IntSet` so that it has all the usual operations, but now mutation is also allowed:

```
/** Represents a **mutable** collection of integers. */
public interface MutableIntSet extends IntSet {
    /**
     * Adds the given item to the set if not already present.
     * @param n The number to add to the set
     * @modifies obj
     * @effects obj = n :: obj_0  if not contains(obj, n)
     *          obj = obj_0       if contains(obj, n)
     */
    void add(int n);

    /**
     * Removes the given number from the set.
     * @param n The number to remove from the set
     * @modifies obj
     * @effects obj = remove(obj_0, n), where
     *     remove(nil, n)    := nil
     *     remove(m :: L, n) := remove(L, n)       if m = n
     *     remove(m :: L, n) := m :: remove(L, n)  if m /= n
     */
    void remove(int n);

    /**
     * Returns a non-mutable version of this set.
     * @return obj
     */
    IntSet freeze();
}
```

The following utility method creates instances of this interface.

```
public class IntSets {
    /**
     * Returns a set containing all the given values.
     * @param vals The list of values that should be in the set
     * @return dedup(vals), where
     *     dedup(nil)    := nil
     *     dedup(n :: L) := n :: dedup(remove(L, n))
     */
    public static MutableIntSet of(int... vals) {
        // .. somehow creates a MutableIntSet ..
    }
}
```

Task 2 – The Sort Jester [17 pts]

In this problem, we will formalize English descriptions of implementations of these interfaces.

a) The `SortedIntSet` class will implement `IntSet` by storing the elements in an array in sorted order. This is a great choice for `IntSet`, but not `MutableIntSet`. (Why?)

Write out a formal specification of the representation in this class. Define the AF and RI. Write out a specification of the constructor, but do not implement it. The constructor takes an integer array of values as an input. It is fine to use English to state properties such as being sorted and containing no duplicates.

```
public class SortedIntSet implements IntSet {
    // AF:
    // RI:
    private final int[] elems;

    /**
     * Your specification of the constructor goes here...
     */
    public SortedIntSet(int[] vals) { ... }
}
```

There is more than one valid choice for how to specify the constructor. Practice good engineering habits by choosing the *easiest* sensible option. ("Over-engineered" is not a compliment.)

b) The following type represents a list of boolean values:

$$\textbf{type}\ \text{BList} := \text{bnil} \mid \text{bcons}(\mathbb{B}, \text{BList})$$

We will use the shorthand "::", when applied to a BList, to refer to a bcons operation. For example, the expression "true :: false :: bnil" is shorthand for $\text{bcons}(\text{true}, \text{bcons}(\text{false}, \text{bnil}))$. Hopefully, this will not cause confusion with cons on List.

Define the function

$$\text{present} : (\text{BList}) \to \text{List}$$

that takes a list of boolean values and returns the list of *indexes* at which it contains true values. In order to do this, you will need to define a helper function,

$$\text{present-from} : (\text{BList}, \mathbb{N}) \to \text{List}$$

where the second input keeps track of the index you are at in the list as you walk through it.

c) The `BooleanIntSet` class will implement `MutableIntSet` by storing an array of booleans that keeps track of which values are present, where each index in the boolean array corresponds to a value. It will also need to keep track of the size of the set, since that is not efficiently calculable from the array of boolean values.

Write out a formal specification of the representation in this class. Define the fields, AF, and RI. Write out a specification of the constructor, but do not implement it. It should take a parameter indicating the maximum integer value that can be stored in the set. (**Hint**: You may need to use the function you defined in the previous part and other list functions.)

```
public class BooleanIntSet implements MutableIntSet {
    // AF:
    // RI:
    private final _____;
    private _____;

    /**
     * Your specification of the constructor goes here...
     */
    public BooleanInstSet(int max) { ... }
}
```

d) In what way does `BooleanIntSet` **not** actually satisfy the specification of `MutableIntSet`? (Sadly, this sort of thing happens often in real-world programs.) How would we fix it?

Task 3 – Rally the Dups [12 pts]

In this problem, we will finish formalizing some aspects left in English above.

a) Write a formal math definition of sorted : (List) $\to \mathbb{B}$. This should return true if and only if the list is in ascending sorted order. (It does not need to sort the list.)

b) Write a formal math definition of no-dups : (List) $\to \mathbb{B}$. This should return true if and only if the list has no duplicate values. (You may call other list functions.)

c) Write a formal math definition of sorted-no-dups : (List) $\to \mathbb{B}$. This should check both properties hold and do so in *linear* time.

Task 4 – Optional: Don't Lose Your Bool [7 pts]

In this problem, we will use AI to implement the methods of `SortedIntSet` and `BooleanIntSet`.

Create a new Java project. Then, paste in the provided definitions of IntSet and `MutableIntSet` and your definitions of `SortedIntSet` and `BooleanIntSet` from Task 2, leaving all method definitions stubbed out (i.e., proper declarations but with the code missing).

a) Which AI are you using?

b) Prompt the AI to fill in *all the methods* of each class, one at a time. Write out your prompts and the code that the AI produces. State whether the code appears to be correct.

c) Did the AI's implementation of `SortedIntSet` properly take advantage of libraries?

d) What are all the places that should be checked for representation exposure? Do you see representation exposure in any of those places in the code produced by the AI?

e) What are all the places where it would make sense to practice defensive programming? Did the AI include the right checks?

Task 5 – Extra Credit: Wicked Witch of the Test [14 pts]

In this problem, we will consider how to test the AI-generated code from Task 4.
Note: You will need to complete Task 4a and 4b to answer these questions.

a) State the number of test cases necessary to achieve proper coverage of `BooleanIntSet.contains`. Then, give a specific set of inputs and the expected outputs for those cases.

b) State the number of test cases necessary to achieve proper coverage of `BooleanIntSet.elements`. Then, give a specific set of inputs and the expected outputs for those cases.

c) If we call one of the mutators, `add` or `remove`, what do we need to check to verify that the resulting state is correct? If we want out unit tests to not test other methods (other than the constructor), what method should we add to the class to make this possible?

d) State the number of test cases necessary to achieve proper coverage of `BooleanIntSet.add`. Then, give a specific set of inputs and the expected state resulting in those cases.

e) State the number of test cases necessary to achieve proper coverage of `BooleanIntSet.remove`. Then, give a specific set of inputs and the expected state resulting in those cases.