CSE 331
Software Design & Implementation

Spring 2023

Section 10 – Final Review

Administrivia

- Final Exam on Tuesday (6/6) in KNE 110
  - **Lecture B**: 2:30-4:20 pm
  - **Lecture A**: 4:30-6:20 pm
  - Primarily focused on loop reasoning and ADTs
- HW9 due tomorrow at 11:00 pm (6/2)

Subtypes – Review

- Recall that subtypes are **substitutable** for supertype
  - If B is a subtype of A, can send B where A is expected

```
function f(s: A): void { … }
function g(): B { … }

const x: B = 3;
f(x);                // okay

const y: A = g();   // okay
```

- For ADTs, we use this as our definition of subtypes
- For B to be substitutable for A, must satisfy 2 conditions:
  1) B must provide all methods of A
  2) B's corresponding methods must…
     - Accept all inputs that A's does
     - Must also promise everything in A's postcondition
     - i.e., B must have the same or **stronger spec**

Equality – Review

- Often useful / necessary to define your own equals
- **Properties of equals method:**

1) equal(a,a) = T	reflexive
2) equal(a,b) = equal(b,a)	symmetric
3) if equal(a,b) and equal(b,c), then equal(a,c)	transitive

Design Patterns – Review

- **3 categories of patterns:**
  - **Creational:**
    - **Builder:** Object that helps with creation of another object
      - lets you describe what you want bit by bit
      - Good for immutable types
  - **Structural:**
    - **Adaptor:** often needed with nominal typing
      - Design pattern for working around language issue
  - **Behavioral:**
    - **Interpreter:** Collects code for similar objects, spreads apart code for operations
      - Easy to add objects, hard to add methods
    - **Procedural:** Collects code for similar operations, spreads apart code for objects
      - Easy to add methods, hard to add objects

Loop Reasoning – Review

Fill in the missing parts of the implementation of `insert`. Your code must be correct with the **provided invariant.** (You do not need to include a proof, but it must be correct).

```
/**
* Returns the value in A that is the smallest out of all values in A that are larger than x
* @param x A number to compare to the values in A.
* @param A A list of numbers
* @param requires A != null
* @returns the smallest of all values in A larger than x
*/
public int nextLargest(A: number[], x: number): number {
    hasLarger: boolean = ______;
    minLarger: number = _____________;
    i: number = __;

    {{ Inv: minLarger = the min value in A[0…i-1] that is larger than x. If no such value exists, hasLarger = false}}
    while (i < A.length) {








    }
    if (!hasLarger) {
        throw new Error(“nothing smaller”);
    }
    return minLarger;
}
```

Loop Reasoning – Midterm

Remember this definition from the previous midterm:

```
/**
 * Returns a set that includes all the current elements and x also
 * @param x a string to insert into the set (if not already present)
 * @returns obj if contains(obj, x) = T
 *          L   if contains(obj, x) = F
 *     where L = A ++ [x] ++ B with obj = A ++ B (i.e., L is an array
 *     containing the strings from obj with x inserted somewhere)
 */
insert(x: string): StringSet;
```

We will implement it with the following class, whose concrete representation is an array sorted in decreasing order.

```
class ArrayStringSet implements StringSet {

  // RI: elems[j] > elems[j+1] for any 0 <= j < elems.length - 1
  // AF: obj = this.elems
  readonly elems: readonly string[];

  // @requires elems is sorted in decreasing order, with no duplicates
  constructor(elems: readonly string[]) {
    this.elems = elems;
  }

  ...
}
```

Fill in the missing parts of the implementation of `insert`. Your code must be correct with the **provided invariant.** (You do not need to include a proof, but it must be correct).

Loop Reasoning – Midterm

```
insert = (x: string): StringSet => {
    const k = findIndex(this.elems, x);
    if (________________________________) {
        return this;
    }
    // Create an array one longer than this.elems.
    const E: string[] = new Array(this.elems.length + 1);
    // Define A := this.elems[0…k-1]
    let i: number = __;
    // Inv: E[0…i-1] = A[0…i-1]
    while (________) {



    }
    // Now we have E[0…i-1] = A and i = k



    // Now we have E[0…i-1] = A ++ [x] and i = k + 1
    // Define B := this.elems[k…this.elems.length-1]. Thus we have this.elems = A ++ B
    let j: number = __;
    // Inv: E[0 .. i - 1] = A ++ [x] ++ B[0 .. j - 1] and i = k + 1 + j
    while (___________________) {




    }
    return new ArrayStringSet(E);
}
```

Loop Reasoning – Midterm

Remember this definition from the previous midterm:

The following function `findIndex` searches for a string in an array of strings that is promised to be sorted in **decreasing** order. In other words, we are promised that $A[0] \geq A[1] \geq \cdots \geq A[n-1]$, where the ordering of strings is according to >= in TypeScript, (reverse) alphabetical ordering.

```
/**
 * Finds the index where x appears in the given sorted array or where, if
 * it is not in the array, it could be inserted to maintain sorted order.
 * @param A Array of strings in *decreasing* order
 * @param x String to look for in a.
 * @returns an integer k such that A[j] > x for any 0 <= j < k and
 *    x >= A[j] for any k <= j < A.length
 */
function findIndex(A: string[], x: string): number
```

(a) Use reasoning to fill in all blank assertions. The ‘Pi’s should be filled in with forward reasoning and the ‘Qi’s with backwards reasoning

(b) Prove Pi implies Qi for i = 1,2,3

Loop Reasoning – Midterm

The precondition is that `A[j] ≥ A[j + 1]` for any `0 ≤ j < n-1`, where n is A.length

```
let k: number = A.length;
{{ P1: ________ }}
{{ Inv: x ≥ A[j] for and k ≤ j < n and k ≥ 0 }}
while (k !== 0 && x >= A[k-1]) {
    {{ P2: _______________________________________________________ }}
    {{ Q2: ___________________________________________ }}
    k = k – 1;
    {{ _______________________________________ }}
}
{{ P3: _________________________________________________________ }}
{{ Q3: A[j] > x for any 0 ≤ j < k and x ≥ A[j] for any k ≤ j < n }}
return k;
```

ADTs – Review

Suppose we have an implementation of a queue using a list, prove the AF holds after the execution of the function

```
class ArrayQueue {
    // RI: 0 <= front < list.length
    // AF: obj = list[front…list.length-1]
    list: number[];
    front: number = 0;
    // adds element to end of queue
    // @effects obj = obj_0 ++ [x]
    enqueue = (x: number): void => {
        this.list.push(x);
    }
    // removes element from front of queue
    // @effects obj_0 = [x] ++ obj if queue is not empty, obj otherwise
    // @returns x if queue is not empty, -1 otherwise
    dequeue = (): number => {
        let x: number;
        if (this.front < this.list.length) {
            x = this.list[this.front];
            this.front = this.front + 1;
            return x;
        }
        return -1;
    }
}
```

Design Pattern – Review

Choose the name of the design pattern that best matches the description below.

(a) We have a program that uses Complex number objects, but we have two possible implementation of Complex - one uses rectangular coordinates, the other uses Polar. We want the program to be able to select during execution which version to use when a new Complex object is created, and not have that decision fixed when the program is compiled.

(b) We have a complicated object with many configurations options. We would like to organize constructors with 12 parameters to set all of the configurations options all at once.

(c) We have a library function that performs calculations using metric units and we want to use it to implement a function that does the same thing, only with U.S. units.

Subtyping – Review

Suppose the class Point3D is a subtype of Point. Which of the functions of Point3D below properly override the function of Point so that Point3D is still substitutable for Point (circle all that apply)?

```
interface Point {
    setX(x: number): void;
    setY(y: number): void;
    // @requires this.x != x and this.y != y
    distance(x: number, y:number): number;
}
```

(a)

```
interface Point3D extends Point {
    setX(x: number | string): void;
    setY(y: number | string): void;
    setZ(z: number | string): void;
    // @requires this.x != x and this.y != y
    distance(x: number, y: number): number;
}
```

(b)

```
interface Point3D extends Point {
    setX(x: number): void;
    setY(y: number): void;
    setZ(z: number): void;
    // @requires this.x != x and this.y != y
    distance(x: number, y: number): number | string;
}
```

(c)

```
interface Point3D extends Point {
    setX(x: number): void;
    setY(y: number): void;
    setZ(z: number): void;
    @returns distance that is < 10
    distance(x: number, y: number): number;
}
```