CSE 331: Software Design & Engineering

Midterm: Section A 5/19/2023 10:30am

Name: __________________________

UW Email: ____________@uw.edu

This exam contains 9 pages (including this cover page) and 4 problems. Check to see if any pages are missing. Enter all requested information on the top of this page.

Instructions:

• Closed book, closed notes, no cell phones, no calculators.
• You have 50 minutes to complete the exam.
• Answer all problems on the exam paper.
• If you need extra space use the back of a page.
• Problems are not of equal difficulty; if you get stuck on a problem, move on.
• It may be to your advantage to read all the problems before beginning the exam.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>Total:</td>
<td>80</td>
<td></td>
</tr>
</tbody>
</table>
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 \( \geq \) in TypeScript, (reverse) alphabetical ordering.

```javascript
/**
 * 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
```

Suppose that the function returns \( k \). If \( x \) is in the array, then we must have \( A[k] = x \). If \( x \) is not in the array, then we must have \( (k = n \text{ or } k \geq 0) \) and \( A[k] \neq x \).

For example, suppose that \( A \) is the array \(["mouse", "dog", "dog", "cat"]\). Then, the specification above tells us that

- A call to `findIndex(A, "zebra")` would return 0.
- A call to `findIndex(A, "dog")` would return 1 (not 2).
- A call to `findIndex(A, "cat")` would return 3.
- A call to `findIndex(A, "bat")` would return 4.
- A call to `findIndex(A, "kangaroo")` would return 1.
1. (18 points) **Loop, There It Is**

Consider the following code, which claims to implement `findIndex` from the prior page.

The precondition is that $A[j] \geq A[j + 1]$ for any $0 \leq j < n - 1$, where $n$ is `A.length`.

```javascript
let k: number = A.length;
{{ P1: ____________________________ }}
{{ Inv: $x \geq A[j]$ for any $k \leq j < n$ and $k \geq 0$ }}
while (k !== 0 && x >= A[k - 1]) {
    {{ P2: ____________________________ }}
    {{ Q2: ____________________________ }}
    k = k - 1;
    {{ ____________________________ }}
}
{{ P3: ____________________________ }}
{{ Q3: $A[j] > x$ for any $0 \leq j < k$ and $x \geq A[j]$ for any $k \leq j < n$ }}
return k;
```

(a) Use reasoning to fill in all blank assertions above. The ‘$P_i$’s should be filled in with forward reasoning and the ‘$Q_i$’s should be filled in with backward reasoning.

(b) Prove that $P_1$ implies Inv.

(Continued on next page...)

(c) Prove that $P_2$ implies $Q_2$.

(d) Prove that $P_3$ implies $Q_3$. 
2. (18 points) **Give It Your Test Shot**

Fill in the body of the following unit test for `findIndex`. Include comments explaining the test cases, as we did in the coding homework problems.

```javascript
it('findIndex', function() {

    // assert.deepEqual(findIndex([],),
    //                 []);

    // assert.deepEqual(findIndex([],),
    //                 []);

    // assert.deepEqual(findIndex([],),
    //                 []);

    // assert.deepEqual(findIndex([],),
    //                 []);

    // assert.deepEqual(findIndex([],),
    //                 []);

    // assert.deepEqual(findIndex([],),
    //                 []);

    // assert.deepEqual(findIndex([],),
    //                 []);

    // assert.deepEqual(findIndex([],),
    //                 []);

    // assert.deepEqual(findIndex([],),
    //                 []);

    // assert.deepEqual(findIndex([],),
    //                 []);

    // assert.deepEqual(findIndex([],),
    //                 []);

});
```
The remaining problems involve the implementation of the following ADT:

/** An array of strings with no duplicates. */
interface StringSet {

/**
 * 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;

/**
 * Returns the largest string in the set
 * @requires obj.length > 0
 * @returns max(obj), where max is defined on non-empty lists by
 * max([y]) := y
 * max(A ++ [y]) := max(A) if y < max(A)
 * max(A ++ [y]) := y if y >= max(A)
 */
max(): string;
}

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;
}

...
3. (28 points) Run Array! Run Array!

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

```typescript
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] as shorthand.
    let i: number =

    // Inv: E[0 .. i - 1] = A[0 .. i - 1]
    // (so E[0 .. i - 1] stores the first i elements from A)
    while (____________________________________) {

    }

    // Now have E[0 .. i - 1] = A and i = k
    // (so E[0 .. i - 1] now stores all of A)

    // Now have E[0 .. i - 1] = A ++ [x] and i = k + 1
    // (so E[0 .. i - 1] now stores all of A followed by x)

(Continued on next page...)
```
// Now have E[0 .. i - 1] = A ++ [x] and i = k + 1 (from previous page)

// Define B := this.elems[k .. this.elems.length-1] as shorthand.
// With these definitions, 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
// (so E[0 .. i - 1] now stores all of A, followed by x, followed by
// the first j elements of B)
while (____________________________________) {

}

// Now have E[0 .. i - 1] = A ++ [x] ++ B and i = A.length + 1 + B.length,
// which means that E = A ++ [x] ++ B as promised.
return new ArrayStringSet(E);
};
4. (16 points) **Here Array, Gone Tomorrow**

(a) Fill in the implementation of `max` in `ArrayStringSet`.

```typescript
max = (): string => {
    // Your implementation here
};
```

(b) Explain in clear English (or prove formally, if you prefer) why your code above is correct.