CSE 331: Software Design & Engineering — Fall 2023

Midterm: Example

Name: _____

UW Email: _____@uw.edu

This exam contains 10 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.
- The final page, which contains reference material, can be removed but must be turned in with your exam.

Problem	Points	Score
1	20	
2	30	
3	18	
4	12	
Total:	80	

1. (20 points) **Straight Off the At**

The function at(n, S), returning the element at *index* n in the list S, is defined as follows:

func at$(n,$ nil$)$	$:=$ undefined	for any $n : \mathbb{N}$	
at$(0,$ cons$(x, L))$	$:= x$	for any $x : \mathbb{Z}$ and L : List	
at$(n+1,$ cons$(x, L))$	$:=$ at(n, L)	for any $n : \mathbb{N}$, $x : \mathbb{Z}$ and L : List	

The function echo(S), returning a list with each element is repeated twice, is defined as:

func echo(nil)	$:=$ nil	
echo(cons(x, L))	$:=$ cons$(x,$ cons$(x,$ echo$(L)))$	for any $x : \mathbb{Z}$ and L : List

The following two properties of the "at" function will be useful to us:

- **Fact A**: at$(2m,$ echo$(S)) =$ at(m, S) for any $m : \mathbb{N}$ and S : List.
- **Fact B**: at$(2m+1,$ echo$(S)) =$ at(m, S) for any $m : \mathbb{N}$ and S : List.

In this problem, we will prove **Fact A**. Do so by **induction** on S.

Hint: since the definition of "at" has a separate cases for $m = 0$ and $m = n + 1$, you may need to separate your induction step into those case as well.

(More space on the next page. . .)

The last three problems involve the following ADT:

```
/** A cursor is a list of integers. */
export interface IntCursor {
  /** @returns obj */
  values: () => List<number>;

  /** @returns at(n, obj) */
  valueAt: (n: number) => number | undefined;
}
```

This ADT will be implemented in the following class, which represents the result of applying echo to a list of integers without actually computing echo (until the caller wants the full list).

```
class EchoCursor implements IntCursor {
  // AF: obj = echo(this.vals)
  readonly vals: List<number>;  // list before echo-ing

  constructor(vals: List<number>) {
    this.vals = vals;
  }

  // ... methods implemented later ...
}
```

The class will have the following factory function:

```
/** @returns a cursor representing the list echo(vals) */
export const makeEchoCursor = (vals: List<number>): IntCursor => {
  return new EchoCursor(vals);
};
```

2. (30 points) **Rally the Loops**

Consider the following code, which claims to implement the `values` method of `EchoCursor`. (Note that, while you could calculate echo(this.vals) using our "bottom up" template, that is not how this one does it, so we will need to double check it works.)

```
values = ():  List<number> => {
  let R: List<number> = this.vals;
  let S: List<number> = nil;
  {{ ____________________________________________________ }}
  {{ Inv: echo(this.vals) = concat(rev(S), echo(R)) }}
  while (R !== nil) {
    {{ ____________________________________________________ }}
    S = cons(R.hd, cons(R.hd, S));
    {{ ____________________________________________________ }}
    R = R.tl;
  }
  {{ echo(this.vals) = rev(S) }}
  return rev(S);
};
```

(a) Use reasoning to fill in the blank assertions above. You must reason **forward** through the code before the loop and **backward** through the code in the body of the loop.

(b) In this part, we will complete the code review by proving the necessary implications: one before the loop, one after exiting the loop, and one at the top of the loop body.

You must prove each of the necessary facts *by calculation*. However, to make things easier, we will allow you to use a special rule to shorten a calculation as follows:

$$\begin{aligned} &\text{echo(this.vals)} \\ &\quad = \text{concat}(\text{rev}(S), \text{echo}(R)) \quad \textbf{Because I said so} \end{aligned}$$

You can use the "Because I said so" rule **one time** in part (b). Alternatively, if you do all the calculations, we will throw out the calculation with the lowest score.

(Continued on the next page. . .)

i. Prove that the first assertion you filled in implies that **Inv** holds initially.
(The definition of concat is included on the final page of the exam.)

ii. Prove that **Inv** and the fact that $R = \text{nil}$ imply that the postcondition holds. You should use the following fact in your calculation:

- Lemma 2: $\text{concat}(L, \text{nil}) = L$ for any list L

(Continued on the next page. . .)

iii. Prove that **Inv** and the fact that $R \neq \text{nil}$ imply that the assertion you filled in at the top of the body of the loop holds. You should use the following facts in your calculation:

- Lemma 3: $\text{rev}(\text{rev}(L)) = L$ for any list L
- Lemma 4: $\text{concat}(\text{concat}(L, R), S) = \text{concat}(L, \text{concat}(R, S))$ for any lists L, R, S.

3. (18 points) **The Next Test Thing**

Fill in the body of the following unit test for the `values` method from Problem 2. Include comments explaining the test cases, as we did in the coding homework problems.

```
it('values', function() {
  // ____________________________________________________________
  assert.deepStrictEqual(
      makeEchoCursor(_____________________________________).values(),
      ____________________________________________________________);

  // ____________________________________________________________
  assert.deepStrictEqual(
      makeEchoCursor(_____________________________________).values(),
      ____________________________________________________________);

  // ____________________________________________________________
  assert.deepStrictEqual(
      makeEchoCursor(_____________________________________).values(),
      ____________________________________________________________);

  // ____________________________________________________________
  assert.deepStrictEqual(
      makeEchoCursor(_____________________________________).values(),
      ____________________________________________________________);

  // ____________________________________________________________
  assert.deepStrictEqual(
      makeEchoCursor(_____________________________________).values(),
      ____________________________________________________________);
}
```

4. (12 points) **I Smell An At**

In this problem, you will implement the "valueAt" method of `EchoCursor`.

```
// @returns at(n, obj)
valueAt = (n: number): number => {









};
```

(a) Fill in the body of the method so that it satisfies the spec.

Your code may not call any functions other than "at" from Problem 1 (which you can assume has been translated to TypeScript) and the built-in `Math` functions.

(b) Explain, in English, why your implementation is correct.

To get full credit, your explanation must reference the abstraction function (since the spec is in terms of the abstract state, obj) and the two facts noted in Problem 1.

Familiar List Functions

The function len(L) returns the length of the list L:

$$\begin{aligned} \textbf{func } \mathsf{len}(\mathsf{nil}) &:= 0 \\ \mathsf{len}(\mathsf{cons}(x, L)) &:= \mathsf{len}(L) + 1 \qquad \text{for any } x : \mathbb{Z} \text{ and } L : \mathsf{List} \end{aligned}$$

The function rev(L) returns a list containing the values of L in reverse order:

$$\begin{aligned} \textbf{func } \mathsf{rev}(\mathsf{nil}) &:= \mathsf{nil} \\ \mathsf{rev}(\mathsf{cons}(x, L)) &:= \mathsf{concat}(\mathsf{rev}(L), \mathsf{cons}(x, \mathsf{nil})) \qquad \text{for any } x : \mathbb{Z} \text{ and } L : \mathsf{List} \end{aligned}$$

The function concat(S, R) returns a list containing the values of S followed by those of R:

$$\begin{aligned} \textbf{func } \mathsf{concat}(\mathsf{nil}, R) &:= R && \text{for any } R : \mathsf{List} \\ \mathsf{concat}(\mathsf{cons}(x, L), R) &:= \mathsf{cons}(x, \mathsf{concat}(L, R)) && \text{for any } x : \mathbb{Z} \text{ and } L, R : \mathsf{List} \end{aligned}$$