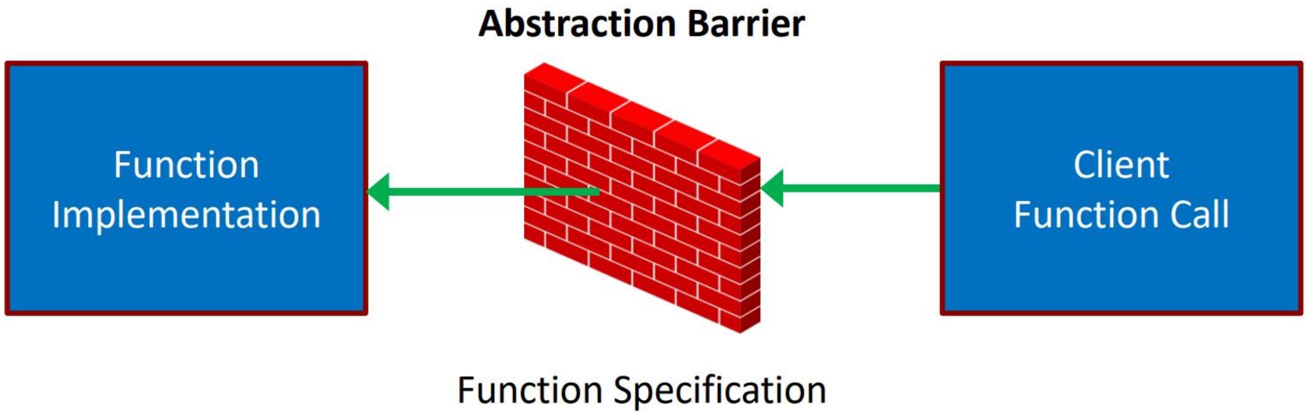
CSE 331
Software Design & Implementation

Spring 2023
Section 4 – Functional Programming III

Administrivia

- HW4 released later today
  - Due Wednesday (4/26) @ 11:00pm
- Deadline to sign up for personal gitlab repos is tonight at 5pm. Please fill out the google form if you want a gitlab repo

Abstraction Barrier – Review



- Specifications acts as the “barrier” between each side
  - This improves understandability, changeability, and modularity
- Clients can only depend on the spec
- Implementer can write any code that satisfies the spec

Defining Interfaces

```
interface FastList {
  getLast(): number|undefined;
  toList(): List<number>;
};
```

Typescript

=

```
interface FastList {
  int getLast() throws EmptyList;
  List<Integer> toList();
}
```

Java

Readonly – Typescript

- The prefix `readonly` is used to make a property as read-only
  - Value cannot be changed
  - Protects variables from unwanted mutations

Ex:

```
class FastLastListImpl extends FastList {
      readonly last: number | undefined;
      readonly list: List<number>;
}
```

Abstract Data Class – Example

```
interface FastList {
  getLast(): number|undefined;
  toList(): List<number>;
};
```

```
class FastLastListImpl extends FastList {
      readonly last: number | undefined;
      readonly list: List<number>;

      constructor (list: List<number>) {
            this.last = last(list);
            this.list = list;
      }
      getLast = () => { return this.last; }
      toList = () => { return this.list; }
}
```

- Can create new record using “new”

```
      new FastLastListImpl(list);
```

Specifications for ADTs – Review

- New Terminology for specifying ADTs
  - Concrete State / Representation (Code)
    - Actual fields of the record and the data stored
    - Ex: `{ list: List, last: number | undefined }`
  - Abstract State / Representation (Math)
    - How clients should think about the object
    - Ex: List (i.e., nil or cons)

Specifications for ADTs – Review

```
/**
 * A list of integers that can retrieve the last
 * element in O(1) time.
 */
export interface FastList {
  …
  /**
   * Returns the object as a regular list of items.
   * @returns obj
   */
 toList(): List<number>
```

We want to give the clients the abstract state specification

- We want to hide the details of the representation from the client (ex: fields are hidden from clients)

Documenting ADTs – Review

Abstract Function (AF) – defines what abstract state the field values currently represent

- Maps the field values to the object they represent
  - Output is math, so this is a mathematical function

Representation Invariants (RI) – facts about the field values that will always be true

- Constructor must always make sure RI is true at runtime
  - Can assume RI is true when reasoning about methods
  - AF only needs to make sense when RI holds
  - Must ensure that RI *always* holds

Documenting ADTs – Review

```
class FastListImpl extends FastList {
  // RI: this.last = last(this.list)
  // AF: obj = this.list
  …
  /** @returns last(obj) */
  getLast(): number | undefined {
    return this.last;
  }
}
```

- **Use both RI and AF to check correctness**

last(obj)	= last(this.list)	**by AF**
	= this.last	**by RI**

Question 4

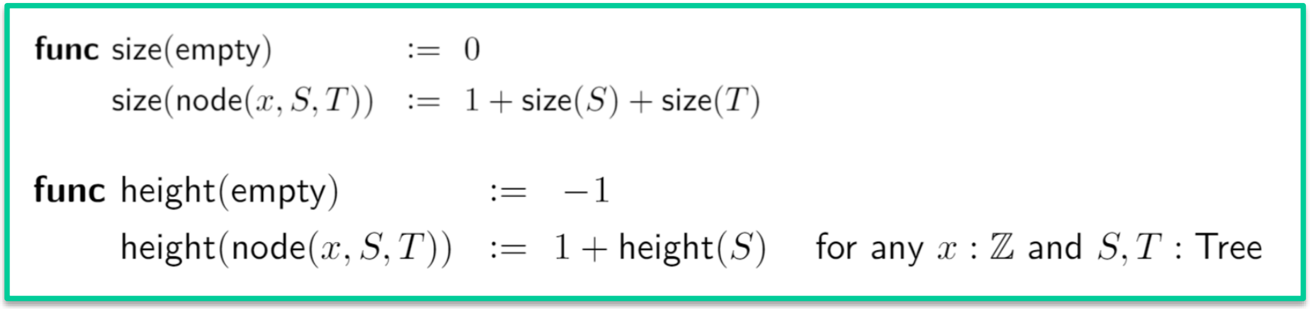
$$\begin{aligned}
&\textbf{func } \mathsf{size}(\mathsf{empty}) &&:= \ 0 \\
&\qquad \mathsf{size}(\mathsf{node}(x, S, T)) &&:= \ 1 + \mathsf{size}(S) + \mathsf{size}(T)
\end{aligned}$$

$$\begin{aligned}
&\textbf{func } \mathsf{height}(\mathsf{empty}) &&:= \ -1 \\
&\qquad \mathsf{height}(\mathsf{node}(x, S, T)) &&:= \ 1 + \mathsf{height}(S) \qquad \text{for any } x : \mathbb{Z} \text{ and } S, T : \mathsf{Tree}
\end{aligned}$$

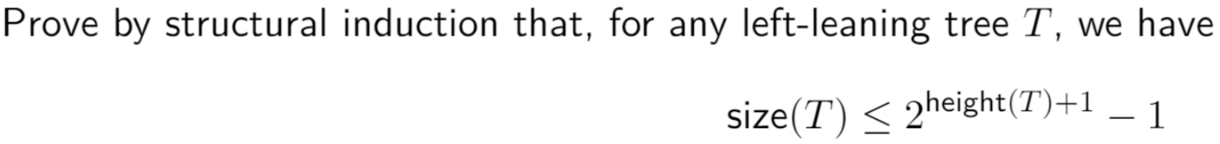
Prove by structural induction that, for any left-leaning tree T, we have

$$\mathsf{size}(T) \leq 2^{\mathsf{height}(T)+1} - 1$$

Question 4



$$\begin{aligned} &\textbf{func } \mathsf{size}(\mathsf{empty}) &&:= 0 \\ &\quad \mathsf{size}(\mathsf{node}(x, S, T)) &&:= 1 + \mathsf{size}(S) + \mathsf{size}(T) \\ \\ &\textbf{func } \mathsf{height}(\mathsf{empty}) &&:= -1 \\ &\quad \mathsf{height}(\mathsf{node}(x, S, T)) &&:= 1 + \mathsf{height}(S) \qquad \text{for any } x : \mathbb{Z} \text{ and } S, T : \mathsf{Tree} \end{aligned}$$



Prove by structural induction that, for any left-leaning tree T, we have

$$\mathsf{size}(T) \leq 2^{\mathsf{height}(T)+1} - 1$$

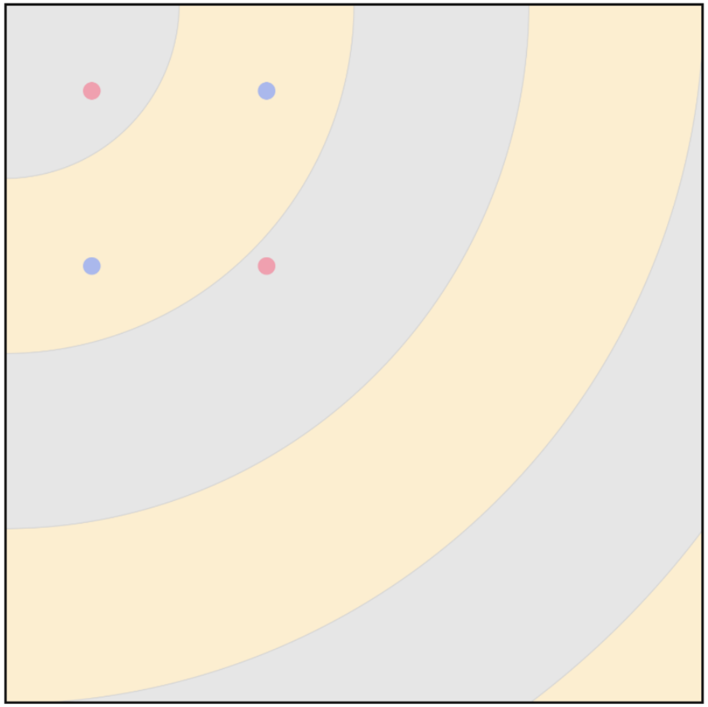
Hint:

1) Define the tree in your IH according to the definition of tree \`node(x, S, T)\` so you can access the left and right trees
2) Remember the exponent rule: $x^y * x = x^{y+1}$

Question 1 & 2 – Coding

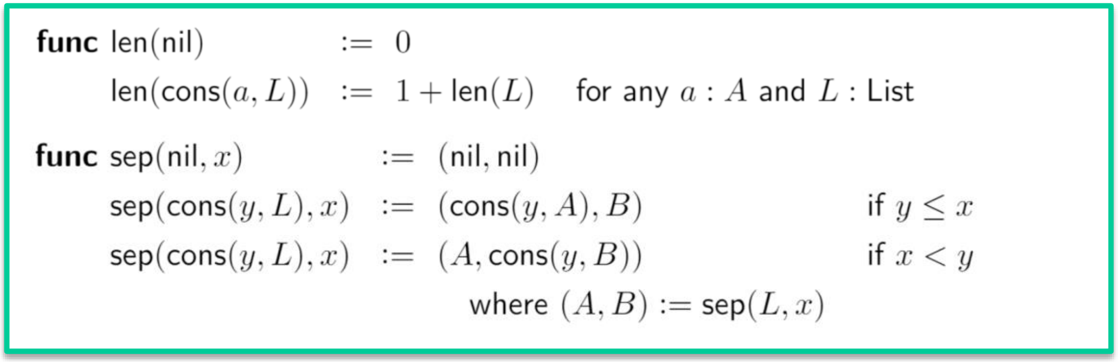
```
git clone git@gitlab.cs.washington.edu:cse331-23sp-materials/sec-highlight.git
```

The application allows the user to type in the coordinates for a list of points and then draws them on a canvas as shown in this picture:

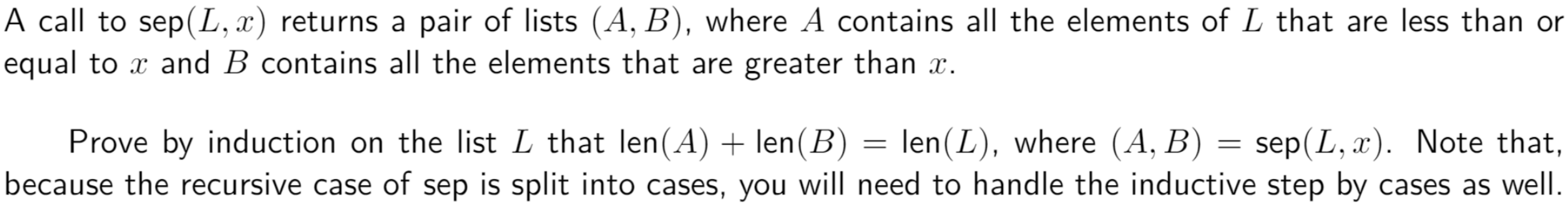


The background of the canvas is striped to show distance from the origin (the upper-left corner). Colors are drawn differently depending on whether they are in a blue or beige stripe.

Question 3



$$\begin{array}{lll}
\textbf{func}\ \mathsf{len}(\mathsf{nil}) & := & 0 \\
\quad\ \mathsf{len}(\mathsf{cons}(a, L)) & := & 1 + \mathsf{len}(L) \quad \text{for any } a : A \text{ and } L : \mathsf{List} \\
\textbf{func}\ \mathsf{sep}(\mathsf{nil}, x) & := & (\mathsf{nil}, \mathsf{nil}) \\
\quad\ \mathsf{sep}(\mathsf{cons}(y, L), x) & := & (\mathsf{cons}(y, A), B) \qquad\qquad \text{if } y \leq x \\
\quad\ \mathsf{sep}(\mathsf{cons}(y, L), x) & := & (A, \mathsf{cons}(y, B)) \qquad\qquad \text{if } x < y \\
 & & \qquad \text{where } (A, B) := \mathsf{sep}(L, x)
\end{array}$$



A call to sep(L, x) returns a pair of lists (A, B), where A contains all the elements of L that are less than or equal to x and B contains all the elements that are greater than x.

Prove by induction on the list L that len(A) + len(B) = len(L), where (A, B) = sep(L, x). Note that, because the recursive case of sep is split into cases, you will need to handle the inductive step by cases as well.

Note: in the recursive case, you make a call to sep(L,x) which then takes the return value of that call (A, B), then finally cons y on to A or B and returns (A, cons(y, B)) or (cons(y, A), B)