UMass Amherst | Manning College of Information & Computer Sciences

Programming Methodology

Lab 4

Wednesday, February 26, 2025

Weekly Lab Agenda

- Go over reminders/goals
- Review past material
- Work in groups of 2-3 to solve a few exercises
    - Please sit with your group from last week.
- Discussion leaders will walk around and answer questions
- Solutions to exercises will be reviewed as a class
- Attendance taken at the end

Reminders

- Homework 4 is due tonight at 11:59pm
    - Come to office hours for help!
- HW5 will not be released until after the exam.
- Midterm 1 is in one week on March 5th (Wednesday)
    - Start studying early. See past exams and solutions on Canvas.
- Midterm 1 Review Session Sunday 7pm - 9pm in ILC TBD

Today’s Goals

- Iterators
- Mental models

Exercise 1: Iterators

- Write a function that takes two iterators over the same type and returns an iterator which will first exhaust iterator 1, then continue with iterator 2.

```
// concatenate two iterators
function concatIt<T>(it1: MyIterator<T>, it2: MyIterator<T>) {
   // your code here
}
```

Exercise 1: Solution

- Write a function that takes two iterators over the same type and returns an iterator which will first exhaust iterator 1, then continue with iterator 2.

```
// concatenate two iterators
function concatIt<T>(it1: MyIterator<T>, it2: MyIterator<T>) {
  return {
    hasNext() { return it1.hasNext() || it2.hasNext() },
    next() { return it1.hasNext() ? it1.next() : it2.next() }
  };
}
```

Exercise 2

For each line of code: If a value is printed, state the value and describe how it was obtained, including any values used for the result. Otherwise, state what objects (including arrays, not closures or functions) are created (if any), what values are modified and which objects are no longer referenced (if any).

```
const mkList = (init, f) => ({
  next: () => mkList(f(init), f),
  value: () => init
});
let cnt = 0;
const a = [mkList(0, x => x + 1), mkList(cnt, _ => ++cnt)];
const b = a.map(lst => lst.next().next());
console.log(b[1].value());
```

Exercise 2 - Solution

5. Variable cnt is declared and initialized to zero.

6. Creates an array with two references to objects returned by mkList

```
a[0] = {
        next: () => mkList((x => x + 1)(0), x => x + 1),
        value: () => 0
}

a[1] = {
        next: () => mkList((_ => ++cnt)(0), _ => ++cnt),
        value: () => 0
}
```

7. For each object lst from a, lst.next().next() is called; mkList is called twice in sequence, creating 2 × 2 objects.

```
1 const mkList = (init, f) => ({
2   next: () => mkList(f(init), f),
3   value: () => init
4 });
5 let cnt = 0;
6 const a = [mkList(0, x => x + 1),
mkList(cnt, _ => ++cnt)];
7 const b = a.map(lst =>
lst.next().next());
8 console.log(b[1].value());
```

Exercise 2 - Solution (Contd.)

7. A new array is created with references to the last object in each sequence.

```
b[0] = {
        next: () => mkList((x => x + 1)(2), x => x + 1),
        value: () => 2
}
b[1] = {
        next: () => mkList((_ => ++cnt)(2), _ => ++cnt),
        value: () => 2
},
```

with cnt = 2, as f is called twice, once for each call to next().

```
1 const mkList = (init, f) => ({
2   next: () => mkList(f(init), f),
3   value: () => init
4 });
5 let cnt = 0;
6 const a = [mkList(0, x => x + 1),
mkList(cnt, _ => ++cnt)];
7 const b = a.map(lst =>
lst.next().next());
8 console.log(b[1].value());
```

Exercise 2 - Solution (Contd.)

8. prints 2, the result of the closure _ => 2, property value of b[1]

Both objects can be garbage collected, they are no longer accessible after executing line 8.

```
1 const mkList = (init, f) => ({
2   next: () => mkList(f(init), f),
3   value: () => init
4 });
5 let cnt = 0;
6 const a = [mkList(0, x => x + 1),
mkList(cnt, _ => ++cnt)];
7 const b = a.map(lst =>
lst.next().next());
8 console.log(b[1].value());
```

Exercise 3: More Iterators

- Write a function that prepends a value to an iterator. That is, return an iterator that will first produce the given value, then continue with the given iterator.

```
function prependIt<T>(v: T, it2: MyIterator<T>): MyIterator<T> {
     // TODO: Complete this function
}
```

Exercise 3: Solution

```
export function prependIt<T>(v: T, it2: MyIterator<T>): MyIterator<T> {
let firstSeen = true;
return {
  hasNext: () => firstSeen || it2.hasNext(),
  next: () => {
    if(firstSeen) {
      firstSeen = false
      return v
    }
    return it2.next()
  }
}
```

Bonus Problem 4: Lists

Consider the following code fragment working with lists as defined in class.
How many list nodes (created with node()) are no longer accessible at the end of this code fragment?

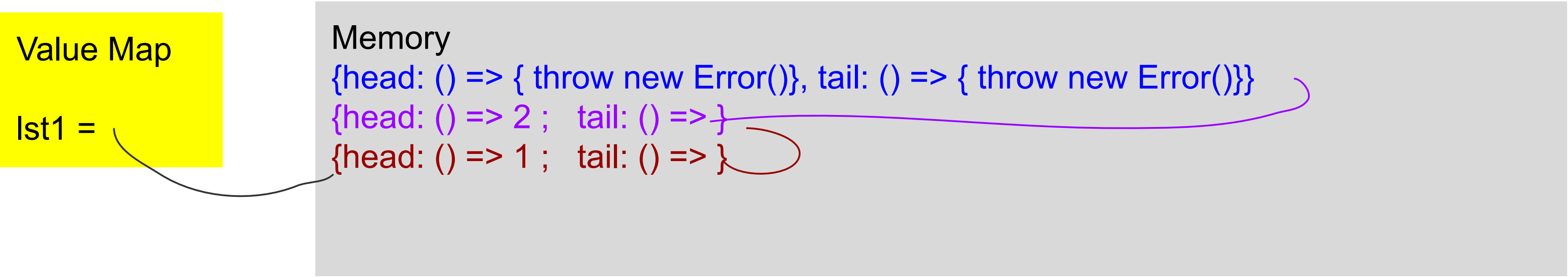
```
let lst1 = ... // create a list with 2 elements
const concat =
  (l1, l2) => l1.isEmpty()? l2 : node(l1.head(), concat(l1.tail(), l2));
lst1 = concat(concat(lst1, lst1), lst1)
// end of the code fragment
```

Hints:
node constructor creates an object of of type List<T> and returns a reference to it.
Every call to the node constructor or to empty() creates a new object in memory.
At the end of the code, we have one variables in value map: lst1.
Objects that are not accessible through lst1 are no longer accessible.

Bonus Problem 4: Lists

Take a List<number> for example. Line 1 has two calls to the node constructor and one call to empty. This creates three objects of type List<number> in memory.

```
let lst1 = node(1, node(2, empty()));
const concat =
  (l1, l2) => l1.isEmpty() ? l2 : node(l1.head(), concat(l1.tail(), l2));
lst1 = concat(concat(lst1, lst1), lst1);
```



Value Map

lst1 =

Memory

{head: () => { throw new Error()}, tail: () => { throw new Error()}}

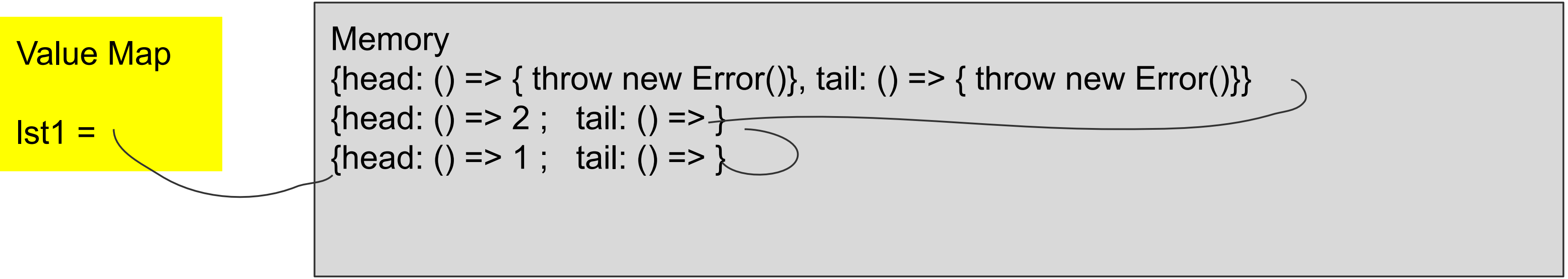
{head: () => 2 ;   tail: () => }

{head: () => 1 ;   tail: () => }

Bonus Problem 4: Lists

Line 2 is a function definition, memory remains the same.

```
let lst1 = node(1, node(2, empty()));
const concat =
  (l1, l2) => l1.isEmpty() ? l2 : node(l1.head(), concat(l1.tail(), l2));
lst1 = concat(concat(lst1, lst1), lst1);
```



Value Map

lst1 =

Memory

{head: () => { throw new Error()}, tail: () => { throw new Error()}}

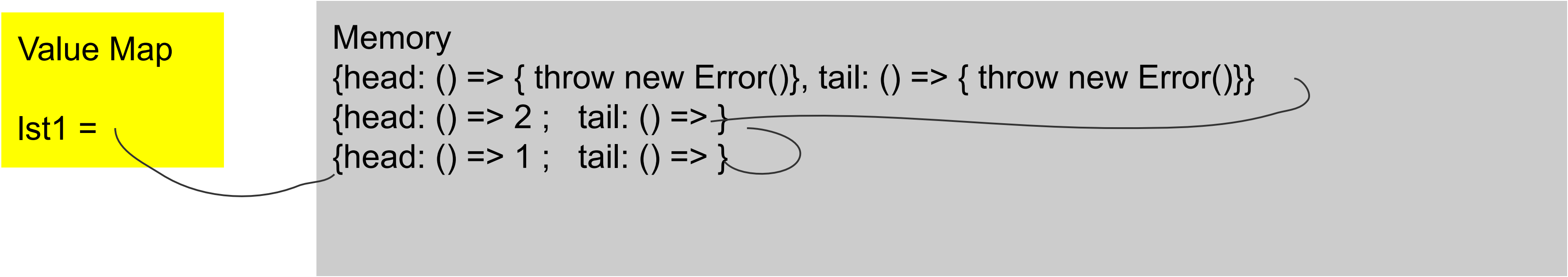
{head: () => 2 ;   tail: () => }

{head: () => 1 ;   tail: () => }

Bonus Problem 4: Lists

Line 3 has two function calls to concat which need to be evaluated to determine the reference that gets assigned to lst1.

```
    let lst1 = node(1, node(2, empty()));
    const concat =
      (l1, l2) => l1.isEmpty() ?  l2 : node(l1.head(), concat(l1.tail(), l2));
    lst1 = concat(concat(lst1, lst1), lst1);
```



Value Map

lst1 =

Memory

{head: () => { throw new Error()}, tail: () => { throw new Error()}}

{head: () => 2 ;   tail: () => }

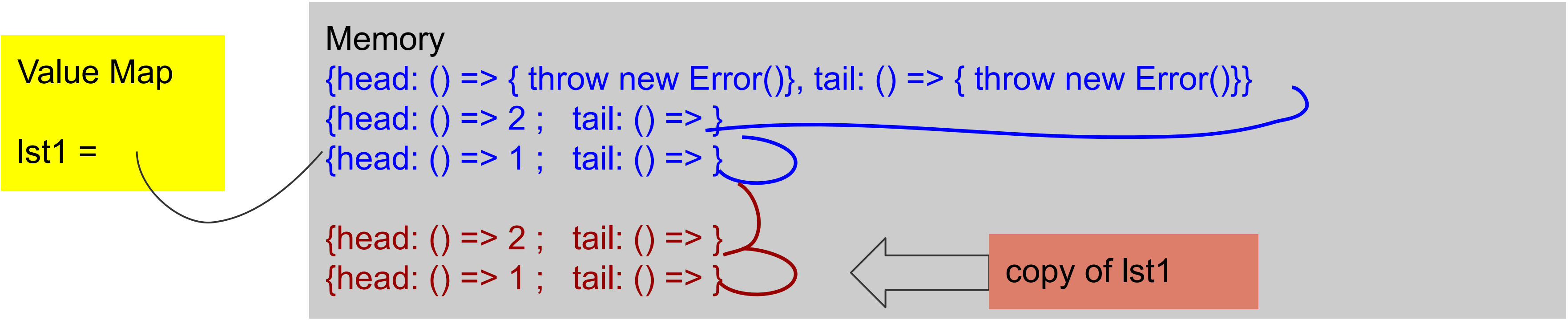
{head: () => 1 ;   tail: () => }

Bonus Problem 4: Lists

Inner call to concat calls the node constructor twice. Two more objects are created, after which l1.isEmpty() evaluates to true. Second object created references the old lst1 through the tail() call.

All objects are accessible through lst1. Note that lst1 hasn’t changed.

```
    let lst1 = node(1, node(2, empty());
    const concat =
      (l1, l2) => l1.isEmpty() ? l2 : node(l1.head(), concat(l1.tail(), l2));
    lst1 = concat(concat(lst1, lst1), lst1);
```



Value Map

lst1 =

Memory

{head: () => { throw new Error()}, tail: () => { throw new Error()}}

{head: () => 2 ;   tail: () => }

{head: () => 1 ;   tail: () => }

{head: () => 2 ;   tail: () => }

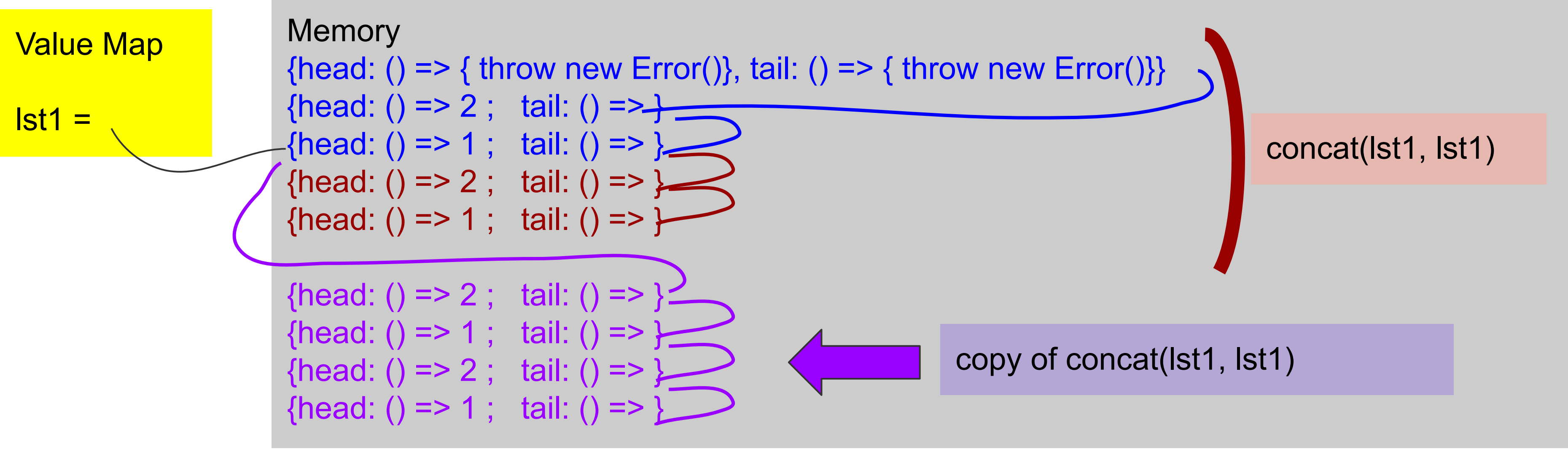
{head: () => 1 ;   tail: () => }

copy of lst1

Bonus Problem 4: Lists

Outer call to concat creates 4 more objects because the first argument concat(lst1, lst1) is a 4-element list. The tail to this 4 element list is set to the reference stored in lst1.

```
    let lst1 = node(1, node(2, empty());
    const concat =
      (l1, l2) => l1.isEmpty() ? l2 : node(l1.head(), concat(l1.tail(), l2));
    lst1 = concat(concat(lst1, lst1), lst1);
```



Value Map

lst1 =

Memory

{head: () => { throw new Error()}, tail: () => { throw new Error()}}

{head: () => 2 ; tail: () => }

{head: () => 1 ; tail: () => }

{head: () => 2 ; tail: () => }

{head: () => 1 ; tail: () => }

{head: () => 2 ; tail: () => }

{head: () => 1 ; tail: () => }

{head: () => 2 ; tail: () => }

{head: () => 1 ; tail: () => }

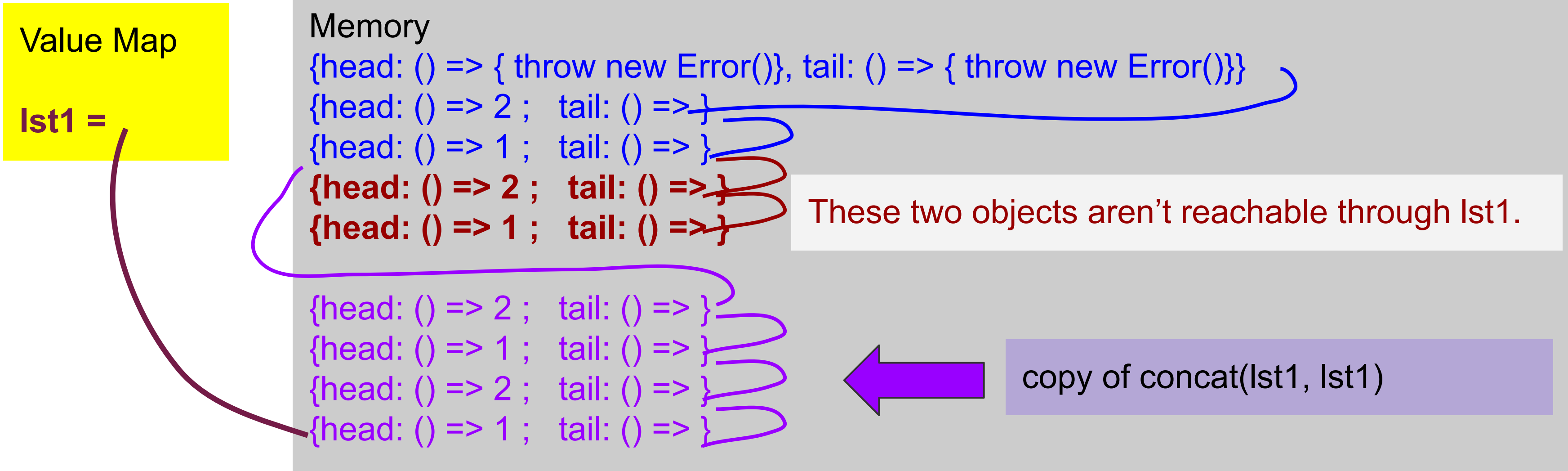
concat(lst1, lst1)

copy of concat(lst1, lst1)

Bonus Problem 4: Lists

After right hand side of the assignment is evaluated, lst1 is updated. Two list nodes are no longer accessible.

```
    let lst1 = node(1, node(2, empty());
    const concat =
      (l1, l2) => l1.isEmpty() ? l2 : node(l1.head(), concat(l1.tail(), l2));
    lst1 = concat(concat(lst1, lst1), lst1);
```



Value Map

lst1 =

Memory

{head: () => { throw new Error()}, tail: () => { throw new Error()}}

{head: () => 2 ;   tail: () => }

{head: () => 1 ;   tail: () => }

{head: () => 2 ;   tail: () => }

{head: () => 1 ;   tail: () => }

These two objects aren’t reachable through lst1.

{head: () => 2 ;   tail: () => }

{head: () => 1 ;   tail: () => }

{head: () => 2 ;   tail: () => }

{head: () => 1 ;   tail: () => }

copy of concat(lst1, lst1)