Programming Methodology

Lab 4

Wednesday, September 27, 2023

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 3 is due tonight at 11:59pm
    - Come to office hours for help!
- HW4 will not be released until after the exam.
- Midterm 1 is in one week on October 4th
    - Start studying early
- Midterm Review Session
    - Sunday (10/1) 7-9 PM @ILC (Room TBD)
    - Will go over Fall 22 Midterm 1 (posted on Canvas)

Today's Goals

- Mental models
- Closures

Exercise 1

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).

```
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 1 - 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 1 - 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 1 - 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 2: Closures

- Write a function that takes as argument an array of Boolean functions, all with the same argument type T and returns a Boolean closure with an argument x of type T. The closure should return true if and only if more than half of the functions in the array return true for x.
- Use reduce for implementation.

```
function mostTrue<T>(
    funarr: ((arg: T) => boolean)[]
): (arg: T) => boolean {
    // TODO
}
```

Exercise 2: Solution

- Write a function that takes as argument an array of Boolean functions, all with the same argument type T and returns a Boolean closure with an argument x of type T. The closure should return true if and only if more than half of the functions in the array return true for x.
- Use reduce for implementation.

```
function mostTrue<T>(
    funarr: ((arg: T) => boolean)[]
): (arg: T) => boolean {
    return (x: T) => funarr.reduce((acc, f) => f(x) ? acc+1 : acc-1, 0) > 0;
}
```

Exercise 3: More closures

- Write a function with no arguments that returns a closure with no arguments. When called the nth time (n >= 1), the closure should return the nth approximation for the number e: 1 + 1/1! + 1/2! + ... + 1/n!
- Avoid needless recomputation in the factorial and in the sum.
- Example outputs: 2, 2.5, 2.666..., 2.70833..., 2.7166..., 2.718055..., etc.

Exercise 3: Solution

- Write a function with no arguments that returns a closure with no arguments. When called the nth time (n >= 1), the closure should return the nth approximation for the number e: 1 + 1/1! + 1/2! + ... + 1/n!
- Avoid needless recomputation in the factorial and in the sum.
- Example outputs: 2, 2.5, 2.666..., 2.70833..., 2.7166..., 2.718055..., etc.

```
function approxE(): () => number {
    let n = 1, factorial = 1, res = 1;
    // can you shorten this even further?
    return () => {
        factorial *= n++; // why n++ and not ++n?
        return (res += 1/factorial);
    }
}
```