UMassAmherst | Manning College of Information
& Computer Sciences

Programming Methodology

Lab 3

Wednesday, February 19, 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 3 is due tonight at 11:59pm, with a free late day (Thursday)
  - Come to office hours for help!
- Homework 4 will be released at latest on Thursday
- If you need to miss lab and have a valid reason according to the syllabus (medical, other personal) please fill out the questionnaire on Canvas before the start time of your lab.
  - Waking up late, bus was late are NOT valid reasons to miss lab.

Today’s Goals

- Closures
- Iterators

Exercise 1: 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 1: 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 2

Consider the code:

```
for (let i = 1; i < n; ++i)
   for (let j = i; j > 0; --j)
      console.log(i, j)
```

where n is some number.

Write a function `mkIterator(n: number): MyIterator<[number,number]>`
that creates an iterator which will produce pairs [number, number]
in the same order as printed by the given code. Only generate pairs on demand.

Use the definition:

```
interface MyIterator<T> { hasNext: () => boolean; next: () => T }
```

Exercise 2 - Solution

```
function mkIterator(n: number): MyIterator<[number, number]> {
  let i = 1, j = 1;
  return {
    hasNext: () => i < n,
    next: () => {
      const r = [i, j];
      if (--j === 0) j = ++i;
      return r;
    }
  };
}
```

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);
    }
}
```