

UMassAmherst

Manning College of Information
& Computer Sciences

Programming Methodology

Lab 9: Midterm 2 Review

Wednesday, April 9th, 2024



Reminders

- Fill out the CATME Survey by this Friday!!
- Midterm 2 is tonight
 - If your lab is at 12:20, you will take the exam in Thompson 102.
 - All other lab times: ILC N151.
- Advice:
 - Write legibly (reduces risk of misgrading)
 - Check the interfaces on the front page when you have doubts about typing
 - Write the types of function parameters and return values
 - The order of the questions doesn't mean anything

Today's Goals

- Review midterm 2 material

Observer Review

- What: A design pattern in which an observable subject automatically notifies dependent observers of any state changes
- Why: It's everywhere. E.g: GUI updates
- How: Reusable class

```
type Observer<T> = (x: T) => any;

class Observable<T> {
  private observers: Observer<T>[] = []; // Maintain a list of observers

  subscribe(f: Observer<T>) {           // Add an observer to the list
    this.observers.push(f);
  }

  update(x: T) {                         // Notify each observer of update
    this.observers.forEach(f => f(x));
  }
}
```

Exercise 1

Write a function **whenAllUpdate** that takes as input an array of observables (all of the same type) and returns a new observable *r*. This new observable should emit an array of values, where each value is the latest update from one of the input observables, but only after all input observables have emitted at least once since the last time *r* emitted.

```
function whenAllUpdate<T>(oa:Observable<T>[]): Observable<T[]>
```

Exercise 1: Solution

```
function whenAllUpdate<T>(oa:Observable<T>[]): Observable<T[]> {
  const r =newObservable<T[]>
  const waiting = Array<boolean>(oa.length).fill(true); // flags: waiting to update
  let remWait=oa.length;// number remaining to update
  let vals: T[]=[];

  oa.forEach((o,i)=>o.subscribe((x: T) => {
    vals.push(x);
    if(waiting[i]){
      waiting[i]=false;
      if(--remWait===0){
        r.update(vals);
        vals=[];// do not reuse array contents
        waiting.fill(true);
        remWait = oa.length; // reinitialize
      }
    }
  }));
  return r;
}
```

Exercise 2: Mental Models

```
1  const f = obj => () => --obj.z;
2  const g = (obj, z) => --obj.z + z;
3
4  let n = 3;
5  const a = [];
6  const b = [];
7  while (--n >= 0) {
8    const obj = { z: n };
9    a.push({ x: f(obj), y: g(obj, obj.z) });
10   b.push(obj);
11 }
12
13 b.map(o => o.z += 2);
14 [0, 1, 2].forEach(i =>
15   console.log(a[i].x(), a[i].y + b[i].z)
16 );
```

For each line of the code, explain what objects and closures are created, what values are computed and printed, and when objects are no longer accessible.

Exercise 2: Solution

```
1  const f = obj => () => --obj.z;
2  const g = (obj, z) => --obj.z + z;
3
4  let n = 3;
5  const a = [];
6  const b = [];
7  while (--n >= 0) {
8    const obj = { z: n };
9    a.push({ x: f(obj), y: g(obj, obj.z) });
10   b.push(obj);
11 }
12
13 b.map(o => o.z += 2);
14 [0, 1, 2].forEach(i =>
15   console.log(a[i].x(), a[i].y + b[i].z)
16 );
```

	Before g	After g
n=2:	/	
obj:	{ z: 2 }	{ z: 1 }
a[0]:	{ x: () => --obj.z,	
	y: obj.z + obj.z }	

b: [obj]

Exercise 2: Solution

```
1  const f = obj => () => --obj.z;
2  const g = (obj, z) => --obj.z + z;
3
4  let n = 3;
5  const a = [];
6  const b = [];
7  while (--n >= 0) {
8    const obj = { z: n };
9    a.push({ x: f(obj), y: g(obj, obj.z) });
10   b.push(obj);
11 }
12
13 b.map(o => o.z += 2);
14 [0, 1, 2].forEach(i =>
15   console.log(a[i].x(), a[i].y + b[i].z)
16 );
```

n=2:

```
obj: { z: 2 }  obj: { z: 1 }
a[0]: {x: () => --obj.z,
      y: obj.z + obj.z}
```

n=1:

```
obj: { z: 1 }  obj: { z: 0 }
a[1]: {x: () => --obj.z,
      y: obj.z + obj.z}
```

b: [obj, obj]

Exercise 2: Solution

```
1  const f = obj => () => --obj.z;
2  const g = (obj, z) => --obj.z + z;
3
4  let n = 3;
5  const a = [];
6  const b = [];
7  while (--n >= 0) {
8    const obj = { z: n };
9    a.push({ x: f(obj), y: g(obj, obj.z) });
10   b.push(obj);
11 }
12
13 b.map(o => o.z += 2);
14 [0, 1, 2].forEach(i =>
15   console.log(a[i].x(), a[i].y + b[i].z)
16 );
```

```
n=2:
  obj: { z: 2 }  obj: { z: 1 }
    a[0]: {x: () => --obj.z,
           y: obj.z + obj.z}
n=1:
  obj: { z: 1 }  obj: { z: 0 }
    a[1]: {x: () => --obj.z,
           y: obj.z + obj.z}
n=0:
  obj: { z: 0 }  obj: {z: -1}
    a[2]: {x: () => --obj.z,
           y: obj.z + obj.z}
```

```
b: [ obj, obj, obj ]
```

Exercise 2: Solution

```
1  const f = obj => () => --obj.z;
2  const g = (obj, z) => --obj.z + z;
3
4  let n = 3;
5  const a = [];
6  const b = [];
7  while (--n >= 0) {
8    const obj = { z: n };
9    a.push({ x: f(obj), y: g(obj, obj.z) });
10   b.push(obj);
11 }
12
13 b.map(o => o.z += 2);
14 [0, 1, 2].forEach(i =>
15   console.log(a[i].x(), a[i].y + b[i].z)
16 );
```

```
n=2:
  obj: { z: 2 }  obj: { z: 1 }
    a[0]: {x: () => --obj.z,
           y: 1 + 2}
n=1:
  obj: { z: 1 }  obj: { z: 0 }
    a[1]: {x: () => --obj.z,
           y: 0 + 1}
n=0:
  obj: { z: 0 }  obj: {z: -1}
    a[2]: {x: () => --obj.z,
           y: -1 + 0}
```

```
b: [ obj, obj, obj ]
```

Exercise 2: Solution

```
1  const f = obj => () => --obj.z;
2  const g = (obj, z) => --obj.z + z;
3
4  let n = 3;
5  const a = [];
6  const b = [];
7  while (--n >= 0) {
8    const obj = { z: n };
9    a.push({ x: f(obj), y: g(obj, obj.z) });
10   b.push(obj);
11 }                                     Resulting array garbage
12                                     /   collected
13 b.map(o => o.z += 2);
14 [0, 1, 2].forEach(i =>
15   console.log(a[i].x(), a[i].y + b[i].z)
16 );
```

```
n=2:
  obj: { z: 2 }  obj: { z: 3 }
    a[0]: {x: () => --obj.z,
           y: 3}
n=1:
  obj: { z: 1 }  obj: { z: 2 }
    a[1]: {x: () => --obj.z,
           y: 1}
n=0:
  obj: { z: 0 }  obj: { z: 1 }
    a[2]: {x: () => --obj.z,
           y: -1}
```

```
b: [ obj, obj, obj ]
```

Exercise 2: Solution

```

1  const f = obj => () => --obj.z;
2  const g = (obj, z) => --obj.z + z;
3
4  let n = 3;
5  const a = [];
6  const b = [];
7  while (--n >= 0) {
8    const obj = { z: n };
9    a.push({ x: f(obj), y: g(obj, obj.z) });
10   b.push(obj);
11 }
12
13 b.map(o => o.z += 2);
14 [0, 1, 2].forEach(i =>
15   console.log(a[i].x(), a[i].y + b[i].z)
16 );

```

```

n=2:
  obj: { z: 2 }  obj: { z: 2 }
    a[0]: {x: () => --obj.z,
           y: 3}
n=1:
  obj: { z: 1 }  obj: { z: 2 }
    a[1]: {x: () => --obj.z,
           y: 1}
n=0:
  obj: { z: 0 }  obj: { z: 1 }
    a[2]: {x: () => --obj.z,
           y: -1}

```

```

i=0:
  a[0].x(): --3 -> 2
  a[0].y + b[0].z: 3+2=5

```

Exercise 2: Solution

```

1  const f = obj => () => --obj.z;
2  const g = (obj, z) => --obj.z + z;
3
4  let n = 3;
5  const a = [];
6  const b = [];
7  while (--n >= 0) {
8    const obj = { z: n };
9    a.push({ x: f(obj), y: g(obj, obj.z) });
10   b.push(obj);
11 }
12
13 b.map(o => o.z += 2);
14 [0, 1, 2].forEach(i =>
15   console.log(a[i].x(), a[i].y + b[i].z)
16 );

```

```

n=2:
  obj: { z: 2 }  obj: { z: 2 }
    a[0]: {x: () => --obj.z,
           y: 3}
n=1:
  obj: { z: 1 }  obj: { z: 1 }
    a[1]: {x: () => --obj.z,
           y: 1}
n=0:
  obj: { z: 0 }  obj: { z: 1 }
    a[2]: {x: () => --obj.z,
           y: -1}

```

```

i=1:
  a[1].x(): --2 -> 1
  a[1].y + b[1].z: 1+1=2

```

Exercise 2: Solution

```

1  const f = obj => () => --obj.z;
2  const g = (obj, z) => --obj.z + z;
3
4  let n = 3;
5  const a = [];
6  const b = [];
7  while (--n >= 0) {
8    const obj = { z: n };
9    a.push({ x: f(obj), y: g(obj, obj.z) });
10   b.push(obj);
11 }
12
13 b.map(o => o.z += 2);
14 [0, 1, 2].forEach(i =>
15   console.log(a[i].x(), a[i].y + b[i].z)
16 );
```

Array gets garbage collected!

```

n=2:
  obj: { z: 2 }  obj: { z: 2 }
    a[0]: {x: () => --obj.z,
           y: 3}
n=1:
  obj: { z: 1 }  obj: { z: 1 }
    a[1]: {x: () => --obj.z,
           y: 1}
n=0:
  obj: { z: 0 }  obj: { z: 0 }
    a[2]: {x: () => --obj.z,
           y: -1}
```

```

i=2:
  a[2].x(): --1 -> 0
  a[2].y + b[2].z: -1+0=-1
```

Exercise 3: Property Based Testing

Write property-based tests for function `partition(arr: number[], p: number)` with the following specification:

- The array elements and `p` are assumed to be integers between 0 and `arr.length-1`.
- The function rearranges the array in place, so that: a number with value $> p$ will not appear before a number with value $\leq p$, and a number with value $< p$ will not appear after a number with value $\geq p$.

Try to write a complete set of tests. You need not write code, but then describe very clearly for each test what you check and how.

Exercise 3: Solution

Tests:

1. Check that the original and resulting array have the same length
2. Check that they contain the same elements with the same frequencies.
Together, this can be done by making a copy before partition and comparing:
3. Check that the result has elements less than p , then elements equal to p , then elements greater than p (any of those may be missing).

Exercise 3: Solution

Tests:

1. Check that the original and resulting array have the same length
2. Check that they contain the same elements with the same frequencies.
Together, this can be done by making a copy before partition and comparing:
3. Check that the result has elements less than p , then elements equal to p , then elements greater than p (any of those may be missing).

```
function checkSame(a: number[], b: number[]): void {
  assert(a.length === b.length);
  const inRange = e => 0 <= e && e < a.length;
  let f = new Array<number>(a.length).fill(0);
  b.forEach(e => { assert(inRange(e)); ++f[e]; });
  a.forEach(e => assert(--f[e] >= 0));
}
```

```
function checkPivot(a: number[], p: number): void {
  let i = a.length; if (i === 0) { return; }
  while (i > 0 && a[--i] > p) {} //all > p
  while (i >= 0 && a[i] === p) { --i; } //all = p
  while (i >= 0) { assert(a[i] < p); --i; }
}
```

Counts elements




Exercise 4: Streams

Write a function that takes an infinite stream of numbers a_1, a_2, \dots and a positive integer k , and returns a stream of running averages of k values, starting with $\text{average}(a_1, a_2, \dots, a_k)$, $\text{average}(a_2, a_3, \dots, a_{k+1})$, etc

Exercise 4: Solution

```
function runavg(s, k) { //assumes s infinite
  const start = s; //save reference to start of stream
  let sum = 0;
  for (let i = k; --i > 0; s = s.tail()) { sum += s.head(); } //k - 1 times
  function avg(p, s, sum) {
    sum += s.head();
    return snode(sum/k, () => avg(p.tail(), s.tail(), sum-p.head()));
  }
  return avg(start, s, sum);
}
```



Remove value of element k-1 elements back in
the stream from running sum