Please take a minute to give us anonymous feedback on how the course is going so far.

http://tinyurl.com/CS220-s24-f1

http://tinyurl.com/CS220-s24-f1

UMassAmherst | Manning College of Information & Computer Sciences

Programming Methodology

Lab 3

Wednesday February 21, 2024

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 2 is due tonight at 11:59pm
    - Come to office hours for help!
- Homework 3 releases soon.
- Fill out Feedback form on Canvas under Week 3
- Start reviewing for midterm 1!

Today's Goals

- Lists
- Writing recursive functions
- Reduce

Lists - review

Lists are **recursive**
Lists are either

- empty
- or an **element** followed by a **list**

Lists are **immutable**

- No modifying “pointers” to other list elements, changing list values, etc.

Lists are an **abstract data type** with the following methods:

- list.**isEmpty()**
  - returns **True** if the list is empty()
  - returns **False** if the list is **node(data,next)** [an element followed by a list]
- list.**head()**
- list.**tail()**

Exercise 1

Given 2 ordered lists of numbers, merge them such that resulting list is an ordered list (ascending).

```
// merges two ordered lists
function merge(list1: List<number>, list2: List<number>): List<number>
```

Exercise 1 - Solution


```
// merges two ordered lists

function merge(l1: List<number>, l2: List<number>) {
  if (l1.isEmpty()) return l2;
  if (l2.isEmpty()) return l1;
  const h1 = l1.head();
  const h2 = l2.head();
  return h1 < h2                              // How would you write
    ? node(h1, merge(l1.tail(), l2))          // this with if/else?
    : node(h2, merge(l1, l2.tail()));
}
```

Array Destructuring

TS allows us to destructure arrays:

```
let a, b, rest;
[a, b] = [1, 2]; // a = 1 and b = 2
```

What does … do?

```
const c = [1, 2, 3];
const d = [4, 5, 6];
const e = [...c, ...d]; // e = [1, 2, 3, 4, 5, 6]
```

```
[a, b, …rest] = [‘a’, ‘b’, ‘c’, ‘d’, ‘e’];
// a = ‘a’, b = ‘b’, rest = [‘c’, ‘d’, ‘e’];
```

Review of Reduce

```
function reduce<T, U>(
    a: T[],
    f: (acc: U, e: T) => U,
    init: U
): U {
    let result = init;
    for (let i = 0; i < a.length; ++i) {
        result = f(result, a[i]);
    }
    return result;
}
```

Reduce is used to combine array elements with the same function.

Example: Find the product of all elements of an array a = [3, 2, 6, 2, 2, 0]

```
a.reduce((prod, e) => prod * e, 1);
```

Exercise 2

Return the sum of all positive and the sum of all negative numbers from an array.

function sumPositivesAndNegatives(arr: number[]): [number, number]

Exercise 2 - Solution


```
function sumPositivesAndNegatives(arr: number[]): [number, number] {
  return arr.reduce(
    ([positive, negative], curr) =>                    // no effect if curr is 0
       (curr > 0) ? [positive + curr, negative] : [positive, negative + curr],
    [0, 0]
  );
}
```

Alternative with conditionals:

```
Function sumPositivesAndNegatives(arr: number[]): [number, number] {
   return arr.reduce(function ([positives, negatives], curr) {
      if (curr > 0) {
         return [positive + curr, negative];
      } else { return [positive, negative + curr]; }
   }, [0, 0]);
}
```

Exercise 3

Convert an array of elements of type T (`T[]`) to a list of elements of type T (`List<T>`)

function <T>(arr: T[]): List<T>

Exercise 3

Convert an array of elements of type T (`T[]`) to a list of elements of type T (`List<T>`)

```
function arrayToList<T>(arr: T[]): List<T> {
   function arrayToListIdx(arr: T[], currIdx: number): List<T> {
       if (arr.length == currIdx) {
           return empty<T>();
       }
       return node(arr[currIdx], arrayToListIdx(arr, currIdx + 1));
   }
   return arrayToListIdx(arr, 0);
}
```

or…

```
function arrayToList<T>(arr: T[]): List<T> {
   const reducer = (lst: List<T>, x: T) => node(x, lst);
   return arr.reduceRight(reducer, empty < T > ());
}
```

Bonus Exercise 4

Convert a list of elements of type T (`List<T>`) to an array of elements of type T (`T[]`)

function listToArray<T>(list: List<T>): T[]

Bonus Exercise 4

Convert a list of elements of type T (`List<T>`) to an array of elements of type T (`T[]`)

```
function listToArray<T>(list: List<T>): T[] {
    return list.reduce((arr, e) => {
        arr.push(e);
        return arr;
    }, [] as T[]);
}
```