Programming Methodology

Lab 3

Wednesday September 18, 2024

Weekly Lab Agenda

- Go over reminders/goals
- Review past material
- Work in groups of 2-3 to solve a few exercises
    - New Groups!
- 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 is released.
- Start reviewing for midterm 1!

Today's Goals

- Lists
- Reduce

Lists - review

Lists are **recursive**
Lists are either

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

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 two ordered lists, merge them such that the 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()));
}
```

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);
```

Array Destructuring

JavaScript lets us destructure arrays:

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

What does `...` (spread syntax) 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'];
```

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]);
}
```

Bonus Exercise 3

Write a function `reverseFilter` that filters a list based on a predicate and returns the filtered elements in reverse order.

```
function reverseFilter<T>(list: List<T>, filterF: (x: T) => boolean):
List<T>
```

Example:
Input List: -2, -1, 0, 1, 2
filterF: (e) => e >= 0
Output List: 2, 1, 0

Bonus Exercise 3 - Solution


```
function reverseFilter<T>(list: List<T>, filterF: (x: T) => boolean): List<T> {
  return list.reduce((acc, e) => (filterF(e) ? node(e, acc) : acc), empty());
}
```

Or

```
function reverseFilter<T>(list: List<T>, filterF: (x: T) => boolean): List<T> {
  return list.filter(filterF).reduce((acc, e) => node(e, acc), empty())
}
```