CS115 – Lab 6: Deconstructing and Constructing Lists; introducing `lambda`

Spring 2020

Question 1: Divisors

Exercise

Write a function `(divisors n)` that consumes a non-zero `Nat` and returns a list containing the divisors of `n`, in increasing order
For example, the divisors of 12 are $\{1, 2, 3, 4, 6, 12\}$, so `(divisors 12) => (list 1 2 3 4 6 12)`

Question 2: Geometric Sequence

A geometric sequence is a sequence of numbers where each term after the first is found by multiplying the previous one by a fixed, non-zero number called the *common ratio*. For example, the sequence $2, 6, 18, 54, \ldots$ is a geometric progression with common ratio 3.

Exercise

Write a function `(geo-seq start len ratio)` that returns a `(listof Num)` containing the geometric sequence where the first value is `start` (a `Num`), the length of the sequence is `len` (a `Nat`), and the common ratio is `ratio` (a `Num`). `(geo-seq 2 4 3) => (list 2 6 18 54)` `(geo-seq 1.1 5 1.5) => (list 1.1 1.65 2.475 3.7125 5.56875)`

Question 3: Currency Names

You are given a `(listof (list Str Str))` containing pairs of country–currency, e.g.:

```
(define currencies
  (list (list "USA" "USD")    (list "Norway" "NOK") (list "Switzerland" "CHF")
        (list "Japan" "JPY") (list "Canada" "CDN") (list "Germany" "EUR")))
```

Exercise

Write a function `(lookup-currency country C)` that consumes a `Str` and a `(listof (list Str Str))`. It returns the second `Str` of the item in `C` where the first item is `country`.
For example, `(lookup-currency "Germany" currencies) => "EUR"`
`(lookup-currency "Canada" currencies) => "CDN"`
Note: you may assume that exactly one of the items in `C` has `country` as its first value. (Write this as a requirement.)

Hint

Use `filter`. What can you say about the list that `filter` returns?

Question 4: Bubble Sort

One way of sorting a list is to repeatedly swap adjacent out-of-order values in a list, until the list is in order. Each swap increases how sorted it is. After enough swaps, the list will be sorted.

For example, start at the right in `(list 11 5 13 7)`.

- Compare the last two values, 13 and 7. $13 > 7$, so swap the last two values, giving `(list 11 5 7 13)`.
- Compare the second from last pair, 5 and 7. $5 < 7$, so do nothing; still `(list 11 5 7 13)`.

- Compare the third from last pair, 11 and 5. $11 > 5$, so swap these values, giving `(list 5 11 7 13)`.

By "bubbling" through the list once, `(list 11 5 13 7)` became `(list 5 11 7 13)`, which is better sorted. Bubbling this list again gives `(list 5 7 11 13)`, which is sorted.

Exercise

Using `foldr`, write a function `bubble` that consumes a `(listof Num)`, and returns the result of **one pass** of swapping out-of-order items in the list, **starting at the right**.

```
(bubble (list 1 2 3 4)) => (list 1 2 3 4)
(bubble (list 11 5 13 7)) => (list 5 11 7 13)
(bubble (list 2 6 9 7 4 2 5 7)) => (list 2 2 6 9 7 4 5 7)
```

Ex.

In the file that contains `bubble`, type in this program, and test it.

```
;; (bsort L) return L, sorted in increasing order.
;; bsort: (listof Num) -> (listof Num)
;; Examples:
(check-expect (bsort (list 2 6 9 7 4 2 5 7))
              (list 2 2 4 5 6 7 7 9))

(define (bsort L)
  (foldr (lambda (a b) (bubble b))
         L
         (range 0 (length L) 1)))
```

Ex.

Trace the code, and try to figure out why it works.