Topic1 ALF

0. Warm-up: Recall map, filter, foldr, foldl, build-list. Try to implement them yourself without referring to your note: a) using recursion b) implement map, filter, build-list using foldr

You may not use explicit recursion unless otherwise specified. You are allowed to use ALF unless otherwise specified.

1. Implement a function called is-prime? that consumes a natural number n and produces true if the given number is prime; false otherwise.
2. Implement a function called sum-pos that consumes a list of integers and sums up all the positive integers in the list.
3. Implement a function called increasing-lists that consumes a positive integer n and produces a list of n lists of natural numbers, where the i-th list contains the first i + 1 natural numbers.
4. Implement a function called identity-matrix that consumes a positive integer n and produces an identity matrix.
5. Implement a function map-lofn which consumes a (listof Any) and a list of functions. The functions in the consumed list will have the contract Num → Any. map-lofn produces a list of lists, where each sublist contains the result after applying each function from the consumed list to each number in the consumed (listof Any).
6. We will implement a function called (my-andmap? pred lst) that returns #t if all the elements of lst satisfy pred, and #f otherwise; a function called (my-ormap? pred lst) that returns #t if all the elements of lst satisfy pred, and #f otherwise. Do not use andmap (or ormap) anywhere in your answers.
    a. Implement both functions recursively using basic Racket functions.
    b. Implement both functions using a single call to the standard Racket foldr function without using recursion.
    c. Implement my-andmap? function using my-ormap?; implement my-andmap? function using my-ormap?. You can not use recursion or foldr in this question.
7. Write each of the transformations first-n-evens and cumu-sum described below using only abstract list functions.

```
;; Create a list of the first n even Nats, starting from 0
(check-expect (first-n-evens 4) (list 0 2 4 6))

;; Cumulative sum: Produce a list whose i-th element is the
;; sum of the first i elements of input list
(check-expect (cumu-sum empty) empty)
(check-expect (cumu-sum '(1)) '(1))
(check-expect (cumu-sum '(1 2 3 4 5 6)) '(1 3 6 10 15 21))
```

8. Write a function count-squares that consumes a (listof Int) and produces the number of perfect squares in the given list without explicit recursion. A perfect square is an integer that is the square of an integer.
9. Write a function ascending? that consumes a (listof Int) and produces true if the entries of the list appear in a strictly increasing order, and false otherwise. Note that a list with 0 or 1 entries is ascending.