1. Write a function (included? lst1 lst2) that consumes two lists lst1 lst2 and produces true if lst1 is in lst2 and false otherwise.

Examples:
(included? (list 1 3 2) (list 1 2 3)) => true
(included? empty (list 1 true)) => true
(included? (list 3.5 (make-posn 3 5))
  (list 10 false (make-posn 3 5))) => false

2. Write a function (in-occ? val n lst) that consumes two integers val and n and a list of integers lst and produces true if val appears in lst exactly n times and false otherwise.

Examples:
(in-occ? 5 3 (list 1 2 5 0 5 -10 0 5 -1)) => true
(in-occ? 10 2 (list 10 10 10)) => false

3. Write a function (list>? lst1 lst2) that consumes two lists lst1 and lst2 of the same length and produces a boolean based on the following rule:

Example:
(list>? (list 10 5 -4 0) (list 3 -10 -20 5)) => false
because 10 > 3, 5 > -10, -4 > -20, but 0 < 5

4. Write a function (cons-list-posn lst1 lst2) that consumes two lists (listof Num) lst1 and lst2 of the same length and produces a (listof posn) where the x values of the posn will be taken from lst1 and the y values of the posn will be taken from lst2.

Example:
(cons-list-posn (list 3 2) (list -1 0)) =>
  (list (make-posn 3 -1) (make-posn 2 0))

5. Write a function (list-equiv lst1 lst2) that consumes two lists of numbers and produces true if both lst1 and lst2 have the same values. Note: Order does not matter and assume no repetition of values in each list.

Example:
(list-equiv (list 1 3.5 -10) (list 3.5 1 -10)) => true

The following structure definition will be used in question 6, 7, and 8.

(define-struct student (name avg))
(define student1 (make-student "Nisha" 70))
(define student2 (make-student "Bettina" 90))
(define student3 (make-student "Mbabi" 80))
(define student4 (make-student "Judah" 12))

6. Write a function \((\text{merge-students} \ los1 \ los2)\) that consumes two sorted lists of students, sorted by name, and produces a new list by concatenating the contents of \(\los1\) and \(\los2\).

Example:
(merge-students (list student1 student2) (list student3))
=> (list student2 student3 student1)

7. Write a function \((\text{sort-students-by-name} \ los)\) that consumes a list of students \(\los\) and produces the same list of students sorted by name.

Example:
(sort-students-by-name (list student1 student2 student3 student4))
=> (list student2 student4 student3 student1)

8. Write a function \((\text{sort-students-by-avg} \ los)\) that consumes a list of students \(\los\) and produces the same list of students sorted by avg (from smallest to highest).

Example:
(sort-students-by-avg (list student3 student2 student1 student4))
=> (list student4 student1 student3 student2)

9. Write a function \((\text{shopping} \ \text{consumer} \ \text{retailer})\) that consumes two \((\text{listof} \ \text{Str})\) and produces a new list containing items from the consumer’s list that appear on the retailer’s list.

Example:
(shopping (list "cereal" "apples" "bananas" "pop")
  (list "apples" "bananas" "cereal" "milk" "potatoes"))
=>(list "cereal" "apples" "bananas")

10. Write a function \((\text{unique} \ \los1 \ \los2)\) that consumes two \((\text{listof} \ \text{Nat})\) where \(\los1\) must contain sorted non-decreasing numbers between 0 and 4 (inclusive) and \(\los2\) must contain sorted non-decreasing numbers between 5 and 9 (inclusive). \(\los1\) can contain duplicates but \(\los2\) must contain distinct values. \text{unique} will produce a list containing the numbers from \(\los1\) followed by those from \(\los2\), but with all the duplicates removed from \(\los1\).

Example:
(unique (list 0 0 1 2 2 3 3 4 4 4 4) (list 6 7 8 9))
=>(list 0 1 2 3 4 6 7 8 9)

11. Write a function \((\text{my-lcm} \ n1 \ n2)\) that consumes two \text{Nat} and produces the lowest common multiple of \(n1\) and \(n2\).
Example:
(my-lcm 81 3465) => 31185