HOMEWORK #4:

My First Lisp Program

Due Date: Friday, May the 13th, 11:59:59pm

Write a collection of LISP function definitions.

Submission Guidelines:

Submit via '`cssubmit 3500 a 4`' a single file called "**`myhw4.lisp`**". Your file should include your **name**.

Description:

Using **only** the functional forms described in the LISP notes class document, or other forms of this homework, write the following lisp functional forms:

- `(myLast L)`
  Evaluates to the last element of list L.
  eg. `(myLast '(p a e g))` → `g`

- `(myCount X L)`
  Evaluates to the number of occurrences of X in list L.
  eg. `(myCount 'a '(p k a t p a e g))` → `2`

- `(myMember X L)`
  Evaluates to 'true' if X is in list L, 'false' otherwise.
  eg. `(myMember 'a '(p a e g))` → `t`

- `(myPurge L)`
  Evaluates to a list with all elements of L without repetition.
  eg. `(myPurge '(p a c e p c))` → `(a e p c)`

- `(myCommon L1 L2)`
  Evaluates to a list of elements that are common in both lists L1 and L2. Assume L1 and L2 have no repeated elements.
  eg. `(myCommon '(p a e g) '(a q r e))` → `(a e)`

- `(myGen X Y Z)`
  Given integers X, Y and Z, evaluates to the list of increasing integers, between X and Y inclusive, with Z as the increment (or to nil if such list does not exist)
  eg. `(myGen 3 11 1)` → `(3 4 5 6 7 8 9 10 11)`
  eg. `(myGen 4 20 4)` → `(4 8 12 16 20)`
  eg. `(myGen 3 10 3)` → `(3 6 9)`
  eg. `(myGen 4 4 5)` → `(4)`
  eg. `(myGen 11 3 1)` → `()`

- `(myMap F L)`
  Evaluates to the list which results from applying function F to every element of list L.
  eg. `(myMap (lambda (x) (* 2 x)) '(1 2 3 4) )` → `(2 4 6 8)`

- `(myReduce F L)`
  Evaluates to the the results of applying aggregate function F to the elements of L. L will be of size >= 2. F will be a commutative function.
  eg. `(myReduce (lambda (x y) (+ x y)) '(1 2 3 4 5))` → `15`

The functions all start with 'my' to prevent any conflict with existing forms.

Use **only** the functional forms described in the LISP notes class document.
Do **not use** other Common Lisp pre-defined forms.

END.