CptS 355 HW #2
Xiyu Xie

nth
Define a function that returns the nth element of a list (0-based indexing). Assume that the index, n, is at least 0 and smaller than the length of the list.

```
( define (nth l i)
        (cond
                ( (<  i  0)   null)
                ( (null? l)  null)
                ( (= i  0)    (car l))
                (  #t          ( nth (cdr l) (- i 1)))
         )
)
```

repl
Define a function repl (define (repl l i v) ...) that returns a (new) list which is the same as l except that the ith element is v. Again, assume that the index i is at least 0 and smaller than the length of the list.

```
( define ( repl l i v)
        (cond
                ( (null? l)   null)
                ( (< i   0 )   null)
                ( (= i   0)    (cons v (cdr l)))
                (     #t        (cons (car l)  (repl  (cdr l) (- i 1) v)))
        )
)
```

range
Define a LISP function range like the range function in python: (define (range min max) ...) that return a list of integers (min min+1 ... max-1). If min ≥ max return the empty list.

```
( define ( range min max)
        ( cond
                ( ( >= min max)       null)
                (         #t              (cons min (range (+ min 1) max) ))
        )
)
```

filter
This higher-order function takes as its first argument one-argument function, called a predicate, which returns #t or #f, and as its second argument a list. It returns a list of all elements in the second argument that satisfy that predicate. The elements must appear in the result in the same

order that they appear in the original list.

```
( define (filter func l)
        (cond
                (( null? l)          null)
                (( func (car l))   (cons  (car l)  (filter func (cdr l))))
                (   #t                (filter func (cdr l)))
        )
)
```

merge2

Define a function merge2 that merges two lists of integers, each already in ascending order, into a new list that is also in ascending order. The length of the new list is the sum of the lengths of the original lists.

```
(define (merge2 l1 l2)
        (cond
                ( (null? l1) l2)
                ( (null? l2) l1)
                ( ( < (car l1) (car l2))  ( cons (car l1) ( merge2 (cdr l1) l2)))
                (         #t                    ( cons (car l2) ( merge2  l1      (cdr l2))))
        )
)
```

mergeN

Using merge2 and the reduce function defined in class you can now define mergeN which takes a list of lists, each already in ascending order, and returns a new list containing all of the elements in ascending order.

```
(define (mergeN L)
        (cond
                ((null?  L)    null)
                ( #t            (merge2 (car L) (mergeN (cdr L))))
        )
)
```