Tutorat 9 - Solution

Faculté de Génie – EECS
CSI2520 : PARADIGMES DE PROGRAMMATION

Hiver 2017 – Tutorat 9

Use map to define a procedure, transpose, that takes a list of pairs and returns a pair of lists as follows.
(transpose '((a . 1) (b . 2) (c . 3))) <graphic> ((a b c) 1 2 3)
[Hint: ((a b c) 1 2 3) is the same as ((a b c) . (1 2 3)).]

```
(define transpose
  (lambda (ls)
    (cons (map car ls) (map cdr ls))))
```

Define the procedure make-list, which takes a nonnegative integer n and an object and returns a new list, n long, each element of which is the object.
(make-list 7 '()) <graphic> (() () () () () () ())

```
(define make-list
  (lambda (n x)
    (if (= n 0)
        '()
        (cons x (make-list (- n 1) x)))))
```

Write the function shorter without using length (which returns the shorter of its two list arguments, or the first if the two have the same length).

```
(shorter '(a b c d) '(f g h))
'(f g h)
(define shorter?
  (lambda (ls1 ls2)
    (and (not (null? ls2))
         (or (null? ls1)
             (shorter? (cdr ls1) (cdr ls2))))))

(define shorter
  (lambda (ls1 ls2)
    (if (shorter? ls2 ls1)
        ls2
        ls1)))
```

Extraire une sous-liste

```
(sub '(a b c d e f g h) 3 5 0)
'(d e f)
(define (sub L start stop ctr)
; extract elements start to stop into a list
(cond ( (null? L) L)
  ( (< ctr start) (sub (cdr L) start stop (+ ctr 1)))
  ( (> ctr stop) '() )
  (else (cons (car L)
            (sub (cdr L) start stop (+ ctr 1))) ) ) )
```

Diviser une liste en deux

```
(split '(a b c d e))
'((a b) (c d e))
(define (split L)
; division de la liste en 2:
; retourne ((1ere moitié)(2nde moitié))
(let ((len (length L)))
  (cond ((= len 0) (list L L) )
      ((= len 1) (list L '() ))
      (else (list (firstHalf L (/ len 2))
                (lastHalf L (/ len 2)))))))
(define (firstHalf L N)
  (if (= N 0)
     null
     (if (or (= N 1) (< N 2))
       (list (car L))
       ;else
       (cons (car L) (firstHalf (cdr L) (- N 1)))))

(define (lastHalf L N)
  (if (= N 0) L
     (if (or (= N 1) (< N 2))
     (cdr L)
     ;else
     (lastHalf (cdr L) (- N 1)))
     ))
```

----

tri-fusion (merge sort)

```
(mergelists '(1 3 4 7) '(2 5 6 8))
```

```
'(1 2 3 4 5 6 7 8)
(define (mergelists L M)
; supposer L et M déjà triés
(cond ( (null? L) M)
  ( (null? M) L)
  ( (< (car L)(car M)) (cons (car L)
                              (mergelists (cdr L)M)))
  (else (cons (car M) (mergelists L (cdr M)))) ) )
```