Université Nice Sophia-Antipolis
L2 Info
Programmation Fonctionnelle Avancée – session 2
Examen : 11 juin 2018
Durée : 1 heure 30

Prénom _____

Nom _____

Note

Num. Etu.

Tous les documents sont interdits. Les calculatrices et les téléphones portables doivent être rangés ainsi que tout autre matériel à l'exception d'un stylo et d'un effaceur. Un mémo est donné en fin de sujet. Un code mal parenthésé sera considéré comme faux s'il est de plus mal indenté.

Exercice 1 Produit scalaire (5 points)

En utilisant une récurrence enveloppée, définissez une fonction `(produit-scalaire L1 L2)` qui prend en argument deux listes de nombres `L1` et `L2` de même longueur et qui renvoie leur produit scalaire. Par exemple, `(produit-scalaire '(1 2 3 4) '(2 1 3 0))` renvoie $13 = 1 \times 2 + 2 \times 1 + 3 \times 3 + 4 \times 0$

En utilisant des fonctions d'ordre supérieur `apply` et `map`, reprogrammez cette fonction en une ligne, sans utiliser d'appel par récurrence.

.

Exercice 2 Mise en majuscules (5 points)

Indication : Pour cet exercice, vous pourrez vous aider des fonctions décrites dans l'exemple ci-dessous.

```
> (char->integer #\X)
88
> (integer->char 88)
#\X
> (string->list "abc")
(#\a #\b #\c)
> (list->string '())
""
```

Définissez une fonction `(majuscule c)` qui prend en argument un caractère `c` et qui renvoie le caractère `c` mis en majuscule s'il s'agit d'une lettre minuscule, sinon inchangé. Par exemple, `(majuscule #\A)` renvoie `\#A` et `(majuscule #\a)` renvoie `\#A`.

En déduire une fonction `(majuscules s)` qui prend en argument une chaîne de caractères `s` et qui renvoie la chaîne de caractères obtenue en remplaçant toutes les lettres minuscules par des majuscules. Par exemple, `(majuscule "Ave, Cesar!")` renvoie `"AVE, CESAR!"`

Exercice 3 Listes mutables (5 points)

On représente la cellule d'une liste par la structure mutable ci-dessous :

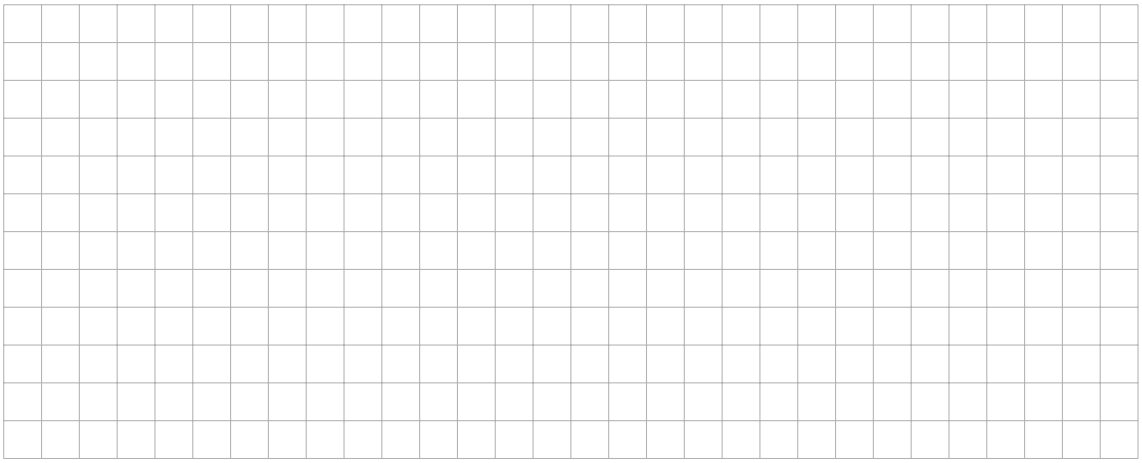
```
(define-struct cell (head tail) #mutable)
(define $empty void)
(define ($empty? L) (equal? L $empty))
(define ($cons x L) (make-cell x L))
(define ($car L) (cell-head L))
(define ($cdr L) (cell-tail L))
(define (set-car! L x) (set-cell-head! L x))
(define (set-cdr! L L2) (set-cell-tail! L L2))
```

.

À l'aide d'une définition récursive, définissez la fonction `mut-list->list` qui transforme une liste mutable en une liste Scheme habituelle.

Par exemple,

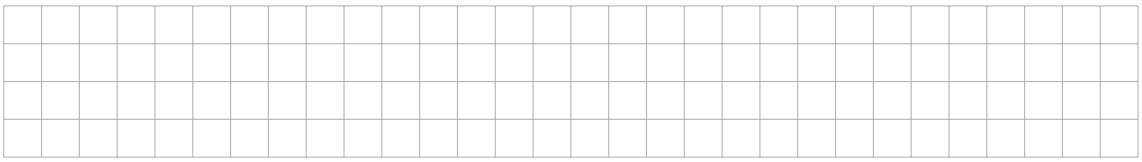
```
> (define L ($cons 1 ($cons 2 ($cons 3 $empty))))
> L
#cell
> (mut-list->list L)
'(1 2 3)
```



À l'aide de la fonction d'ordre supérieur `fold-right`, définissez en une ligne la fonction inverse `list->mut-list` qui convertit une liste Scheme en une liste mutable.

Indication : on rappele sur un exemple le fonctionnement de `fold-right` :

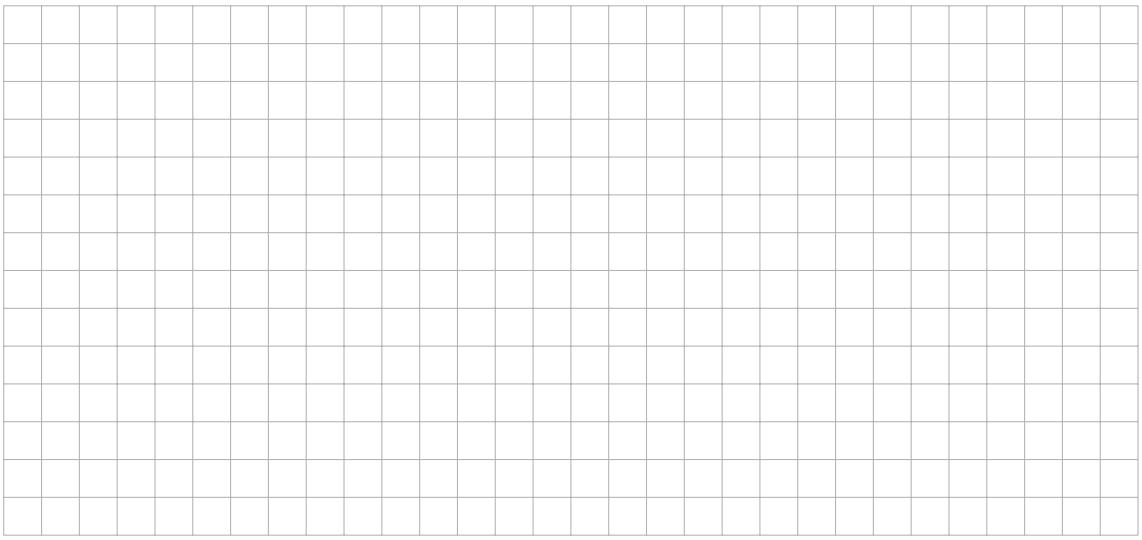
```
> (fold-right - 0 '(1 2 3))
2 ; autrement dit (- 1 (- 2 (- 3 0)))
```



Définissez une fonction `($map f L)` qui prend en argument une fonction `f` et une liste mutable `L` de contenu $L_1, \ldots, L_n$ et qui renvoie une nouvelle liste mutable de contenu $f(L_1), \ldots, f(L_n)$, sans modifier `L`.

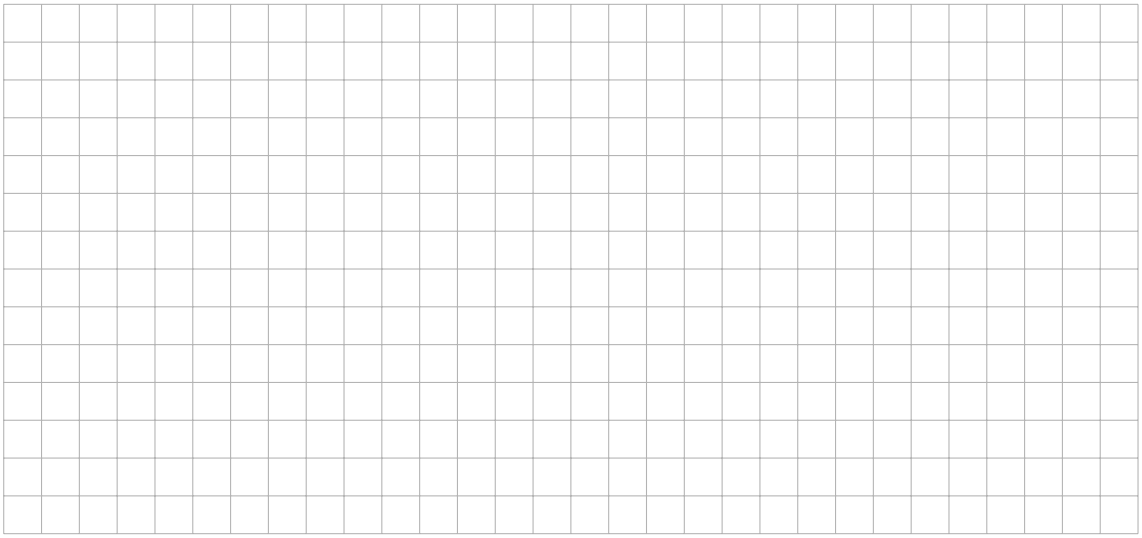
```
> (define L ($cons 1 ($cons 2 ($cons 3 $empty))))
> (define L2 ($map sqr  L))
> (list L L2)
'((1 2 3) (1 4 9))
```

.



Définissez une fonction `($map! f L)` qui prend en argument une fonction `f` et une liste mutable `L` de contenu $L_1, \ldots, L_n$ sans valeur de retour et qui a pour effet de remplacer le contenu de `L` par $f(L_1), \ldots, f(L_n)$.

```
> (define L ($cons 1 ($cons 2 ($cons 3 $empty))))
> ($map! sqr  L)
> L
'(1 4 9)
```



.

Exercice 4 Une file orientée objet (5 points)

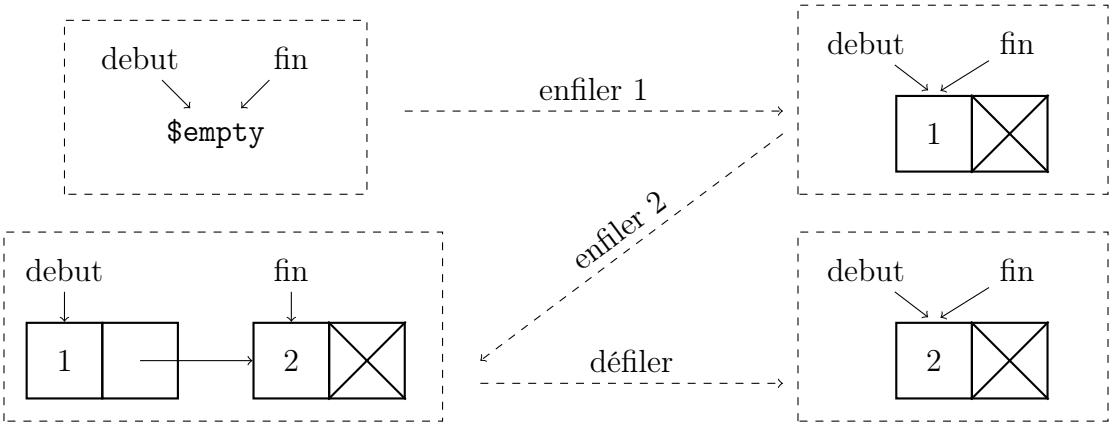
On rappele qu'une file est une structure de données abstraite contenant une liste de valeurs commençant par la plus anciennement enfilée et terminant par la plus récemment enfilée : lorsqu'on enfile une valeur, on la rajoute en fin de liste, et lorsqu'on défile, on retire la première valeur de la liste. On souhaite créer un objet soft qui implémente une file, de sorte que l'on puisse faire par exemple

```
> (define F (nouvelle-file))
> (F 'vide?)
#t
> (F 'enfiler 1)
> (F 'enfiler 2)
> (F 'defiler)
1
> (F 'vide?)
#f
```

En vous aidant du code ci-dessous et de la figure page suivante, définissez un objet soft qui implémente une file.

```
(define (nouvelle-file)
  (local
    [(define debut $empty)
     (define fin $empty)
     (define (enfiler x)
       (if ($empty? fin)
           (begin  ;cas 1: liste vide, on cree une cellule
             (set! debut ($cons x $empty))
             (set! fin debut))
           (begin  ;cas 2: on cree une cellule avec x a la fin
              ...))
      (define (defiler)
       (if ($empty? debut)
           (error "file␣vide")
           (local
                ; la file n'est pas vide, on retire la
                ; premiere cellule en avancant debut.
                ; Si c'etait la seule cellule, on
                ; on met aussi fin a $empty
                [(define res ($car debut))]
                ...)))
      (define (this methode . Largs)
       (case methode
         ((enfiler) ....)
         ((defiler) ....)
         ((vide?) ....)))]
    this)
```

.


debut
fin
$empty
enfiler 1
debut
fin
1
enfiler 2
debut
fin
1
2
défiler
debut
fin
2