Université Bordeaux 1 Licence S&T Mention Informatique S6 2006/2007

UE INF207 - Programmation Fonctionnelle et Symbolique
Devoir surveillé

Tous documents autorisés. Mercredi 28 Mars 2007
Durée : 1h30.
Le barême est donné à titre **indicatif**.
Chacun des quatre exercices doit être **rédigé** sur une feuille **séparée** sur laquelle doit être inscrit votre nom et votre numéro de groupe. Vous devez rendre exactement **une feuille** (ou copie), éventuellement blanche, **par exercice**.

Dans les exercices 2 à 4 il est **obligatoire** *de programmer dans un style* **fonctionnel**, *c'est-à-dire sans utiliser ni* `setf` *ni boucles.*

Exercice 1 *(4pts)*

Évaluer les expressions suivantes :

1. `(append '(1 2) '(3 4))`
2. `(cons '(1 2) '(3 4))`
3. `(cons '(1 2) 3)`
4. `(list 'a (+ 2 3))`
5. `'(a (+ 2 3))`
6. `(mapcar #'max '(2 4 7) '(5 3 8))`
7.
```
(find-if (lambda (x) (zerop (car x)))
         '((1 . a) (0 . b) (2 . c) (0 . d) (3 . e)))
```
8. `(find-if #'evenp '(1 2 3 4 5) :from-end t)`

Exercice 2 *(5pts)*

1. Que fait la fonction `mapfun` suivante :

```
(defun mapfun (funs x)
  (mapcar (lambda (f) (funcall f x)) funs))
```

2. Donner un exemple non trivial d'appel à `mapfun` ainsi que la valeur retournée par cet appel.
3. Écrire la fonction `which (funs x)` qui retourne la liste des éléments `f` de la liste `funs` tels que `(f x)` est vrai. Les éléments de la liste résultat doivent être dans le même ordre que dans la liste passée en paramètre.

Exemple :

```
CL-USER> (which
           (list #'symbolp #'atom #'numberp #'consp #'zerop #'integerp) 0)
(#<FUNCTION ATOM> #<FUNCTION NUMBERP> #<FUNCTION ZEROP> #<FUNCTION INTEGERP>)
```

Exercice 3 *(6pts)*
On peut représenter en *Lisp* une suite de nombres $s = \langle s_0, s_1, s_2, s_3, \ldots \rangle$ par la fonction qui à tout entier naturel i associe le terme s_i. Par la suite, nous appelons *série* une telle suite. La fonction `serie-nth` permet de renvoyer le i-ème terme d'une série :

```
(defun serie-nth (i s)
  (funcall s i))
```

Par exemple, la série $\langle 0, -1, 2, -3, 4, -5, \ldots \rangle$ peut être définie par :

```
(defparameter *alternance*
  (lambda (n)
    (if (evenp n) n (- n))))
```

Exemple d'appel :

```
CL-USER> (serie-nth 43 *alternance*)
-43
```

1. Écrire une fonction `serie-left-shift (s)` qui à toute série `s` associe la série `s` amputée de son premier terme, c'est-à-dire : $\langle s_1, s_2, s_3, \ldots \rangle$.

   Exemple:
   ```
   CL-USER> (serie-nth 43 (serie-left-shift *alternance*))
   44
   ```

2. Écrire une fonction `serie-plus (s u)` qui calcule la somme de deux séries : la somme de deux séries `s` et `u` est la série dont le i-ème terme est la somme du i-ème terme de `s` et du i-ème terme de `u`.
3. Écrire une fonction `serie-eval (s x n)`, où `s` est une série, `x` un nombre réel et `n` un entier naturel, et qui calcule la quantité ci-dessous :

   $$\sum_{i=0}^{i=n-1} s_i \, x^i$$

   Exemple :

   ```
   (defun fact (n)
     (if (zerop n)
         1
         (* n (fact (1- n)))))
   ```

   ```
   CL-USER> (defparameter *E* (lambda (n) (/ 1 (fact n))))
   *E*
   CL-USER> (serie-eval *E* 1.0 11)
   2.7182817
   ```

Exercice 4 *(5pts)*

1. Écrire une fonction récursive `random-partition (n)` ($\texttt{n} \in \mathbb{N}$), qui fabrique aléatoirement une liste d'entiers dans $[1, \texttt{n}]$ et dont la somme est `n`.
   Exemples :

```
CL-USER> (random-partition 0)
NIL
CL-USER> (random-partition 10)
(1 1 2 1 1 4)
CL-USER> (random-partition 10)
(1 8 1)
CL-USER> (random-partition 10)
(2 8)
```

2. Soit `partition`, une partition obtenue en appelant la fonction `random-partition`. La fonction `suivant (partition)` retourne une nouvelle partition. Cette dernière est obtenue en diminuant de 1 chaque entier de `partition`, en ajoutant au début de la liste la somme des 1 retranchés et en supprimant les entiers nuls.
   Exemples :

```
CL-USER> (setf *partition* (random-partition 11))
(1 1 3 2 4)
CL-USER> (suivant *partition*)
(5 2 1 3)
CL-USER> (setf *partition1* (random-partition 10))
(2 5 3)
CL-USER> (suivant *partition1*)
(3 1 4 2)
```

   Écrire la fonction `suivant (partition)`.

Rappel : `random(n)`, où $\texttt{n} > 0$, retourne aléatoirement un entier dans $[0, \texttt{n}[$.