Université Bordeaux 1 Licence S&TS Mention Informatique S5 2010/2011

UE INF353 Programmation 3

Programmation Fonctionnelle et Symbolique

Devoir surveillé No 1

Tous documents autorisés. Jeudi 4 Novembre 2010 Durée : 1h20.
Le barème est donné à titre **indicatif**.
Sur chaque feuille, indiquez vos nom, prénom et numéro de groupe.

Exercice 1 *(2pts)*

Évaluer les expressions suivantes :

1. `(cons '(1 2) '(4))`
2. `(append '(1 2) '(4))`
3. `(list '(1 2) '(4))`
4. `(append '(1 2) '4)`

Exercice 2 *(2pts)*

Combien de paires sont-elles nécessaires pour représenter de manière interne les structures affichées par le `Printer` ?

1. `((1 . 2) 3 . 4)`
2. `((1 2) (3 4))`

Exercice 3 *(10pts)*

Soient les deux fonctions `mystere1` et `mystere2` données en annexe page 3.

1. Que retourne l'appel `(mystere1 '(1 2 3 4 5) #'oddp)` ?
2. Que retourne l'appel `(mystere2 '(1 2 3 4 5) #'oddp)` ?
3. Donner un autre exemple d'appel à `mystere2` ainsi que la valeur de retour de cet appel.
4. `mystere1` et `mystere2` sont-elles équivalentes (même effets pour mêmes arguments) ?
5. Décrire en une phrase la fonction `mystere2` (ce qu'elle prend en entrée et ce qu'elle retourne en sortie).

Soient les deux appels présentés Fig. 1 en annexe page 3.

6. Expliquer la différence de comportement entre ces deux appels.
7. Écrire une version récursive terminale de la fonction `mystere2`.

Exercice 4 *(6pts)*

Un 2-*ensemble* est un ensemble où les éléments peuvent apparaître avec une multiplicité d'au plus 2, c'est à dire 0, 1 ou 2 fois.

1. Écrire une fonction `list-to-2set (l)` qui transforme une liste d'atomes quelconques en un 2-ensemble en supprimant les éléments qui apparaissent déjà deux fois. l'**ordre** des éléments n'est pas important. On pourra utiliser la fonction `count` dont la documentation `HyperSpec` est donnée Fig. 2 en annexe page 4.

   Exemples :

   ```
   CL-USER> (list-to-2set '(1 1 1 2 2 3))
   (1 1 2 2 3)
   CL-USER> (list-to-2set '(1 2 1 3 2 1 2 4 2 1 3))
   (3 1 2 4 2 1 3)
   ```

2. Étendre la fonction précédente de manière à pouvoir traiter des 2-ensembles de n'importe quelle nature en passant en *paramètre mot-clé* le test à utiliser pour comparer les éléments. Exemple d'utilisation :

   ```
   CL-USER> (list-to-2set '((1 0) (2) (1 0) (2) (2) (1 0)) :test #'equal)
   ((1 0) (2) (2) (1 0))
   ```

FIN

```
(defun mystere1 (l f)
  (if (endp l)
      l
      (let ((rest (mystere1 (cdr l) f)))
        (if (funcall f (car l))
            (append rest (list (car l)))
            rest))))

(defun mystere2 (l f)
  (labels ((aux (l)
             (if (endp l)
                 l
                 (let ((rest (aux (cdr l))))
                   (if (funcall f (car l))
                       (cons (car l) rest)
                       rest)))))
    (nreverse (aux l))))


CL-USER> (defparameter *l* (make-list 10000 :initial-element 1))
*L*
CL-USER> (time (defparameter *res* (mystere1 *l* #'identity)))
Evaluation took:
  4.326 seconds of real time
  1.096068 seconds of user run time
  0.340021 seconds of system run time
  [Run times include 0.368 seconds GC run time.]
  0 calls to %EVAL
  0 page faults and
  800,501,184 bytes consed.
*RES*
CL-USER> (time (defparameter *res* (mystere2 *l* #'identity)))
Evaluation took:
  0.001 seconds of real time
  0.004001 seconds of user run time
  0.0 seconds of system run time
  0 calls to %EVAL
  0 page faults and
  159,744 bytes consed.
*RES*
```

FIGURE 1 – Fonctions `mystere1` et `mystere2` et exemples d'exécution

```
Function COUNT, COUNT-IF, COUNT-IF-NOT

Syntax:
count item sequence &key from-end start end key test test-not => n
count-if predicate sequence &key from-end start end key => n
count-if-not predicate sequence &key from-end start end key => n

Arguments and Values:

item---an object.

sequence---a proper sequence.

predicate---a designator for a function of one argument that returns
            a generalized boolean.
from-end---a generalized boolean. The default is false.

test---a designator for a function of two arguments that returns
       a generalized boolean.
test-not---a designator for a function of two arguments that returns
           a generalized boolean.
start, end---bounding index designators of sequence.
             The defaults for start and end are 0 and nil, respectively.
key---a designator for a function of one argument, or nil.

n---a non-negative integer less than or equal to the length of sequence.

Description:

count, count-if, and count-if-not count and return the number of elements
in the sequence bounded by start and end that satisfy the test.

The from-end has no direct effect on the result.
However, if from-end is true, the elements of sequence will be supplied
as arguments to the test, test-not, and key in reverse order, which may
change the side-effects, if any, of those functions.

Examples:
 (count #\a "how many A's are there in here?") =>  2
 (count-if-not #'oddp '((1) (2) (3) (4)) :key #'car) =>  2
 (count-if #'upper-case-p "The Crying of Lot 49" :start 4) =>  2
```

FIGURE 2 – Documentation HyperSpec de la fonction count